(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 0 713 311 A1

(12) EUROPEAN PATENT APPLICATION

(43) Date of publication:
22.05.1996 Bulletin 1996/21

(51) Int Cl.[6]: **H04L 29/06**

(21) Application number: **95308261.7**

(22) Date of filing: **17.11.1995**

(84) Designated Contracting States:
DE FR GB IT

(30) Priority: **18.11.1994 CA 2136150**

(71) Applicant: **Milkyway Networks Corporation**
Ottawa, Ontario K2B 8H6 (CA)

(72) Inventor: **Vu, Hung T.**
Ottawa, Ontario K2B 5X1 (CA)

(74) Representative: **Jones, Graham H.**
Graham Jones & Company
77 Beaconsfield Road
Blackheath
London SE3 7LG (GB)

(54) Secure gateway and method for communication between networks

(57) An apparatus and method for providing a secure firewall between a private network (10) and a public network (12) are disclosed. The apparatus is a gateway station (14) having an operating system that is modified to disable communications packet forwarding, and further modified to process any communications packet having a network encapsulation address which matches the device address of the gateway station. The method includes enabling the gateway station to transparently initiate a first communications session with a client on a first network requesting a network service from a host on a second network, and a second independent communications session with the network host to which the client request was addressed. The data portion of communications packets from the first session are passed to the second session, and vice versa, by application level proxies which are passed the communications packets by the modified operating system. Data sensitivity screening is preferably performed on the data to ensure security. Only communications enabled by a security administrator are permitted. The advantage is a transparent firewall with application level security and data screening capability.

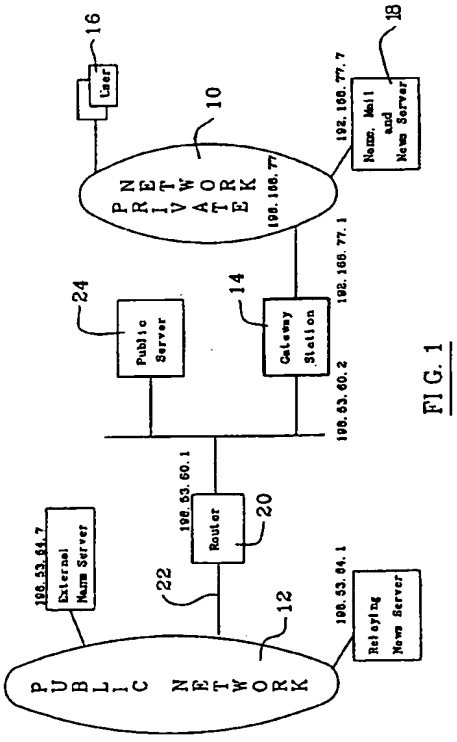

FIG. 1

EP 0 713 311 A1

Printed by Jouve, 75001 PARIS (FR)

Description

TECHNICAL FIELD

This application relates generally to internetwork communications and data exchanges and, in particular, to secure gateways which serve as firewalls between computer networks to inhibit electronic vandalism and espionage.

BACKGROUND OF THE INVENTION

As computing power and computer memory have been miniaturized and become more affordable, computer networks have largely displaced mainframe and minicomputer technology as a business automation platform. Public information networks have also sprung up around the world. The largest and most pervasive public network is the Internet which was created in the late 1960s as a United States Department of National Defence project to build a network connecting various military sites and educational research centers. While the interconnection of private networks with public networks such as the Internet may provide business opportunities and access to vital information, connecting a private, secure network to a public network is hazardous unless some form of secure gateway is installed between the two networks to serve as a "firewall".

Public networks, as their name implies, are accessible to anyone with compatible hardware and software. Consequently, public networks attract vandals as well as amateurs and professionals involved in industrial espionage. Private networks invariably store trade secret and confidential information which must be protected from exposure to unauthorized examination, contamination, destruction or retrieval. Any private network connected to a public network is vulnerable to such hazards unless the networks are interconnected through a secure gateway which prevents unauthorized access from the public network.

A great deal of effort has been dedicated to developing secure gateways for internetwork connection. As noted above, these gateways are commonly referred to as firewalls. The term firewall is broadly used to describe practically any internetwork security scheme. Firewalls are generally developed on one or more of three models: the screening router, the bastion host and the dual homed gateway. These models may be briefly defined as:

Screening router - Screening routers typically have the ability to block traffic between networks or specific hosts on an IP port level. Screening routers can be specially configured commercial routers or host-based packet filtering applications. Screening routers are a basic component of many firewalls. Some firewalls consist exclusively of a screening router or a packet filter.

Bastion host - Bastion hosts are host systems positioned between a private network and a public network which have particular attention paid to their security. They may run special security applications, undergo regular audits, and include special features such as "sucker traps" to detect and identify would-be intruders.

Dual homed gateway - A dual homed gateway is a bastion host with a modified operating system in which TCP/IP forwarding has been disabled. Therefore, direct traffic between the private network and the public network is blocked. The private network can communicate with the gateway, as can the public network but the private network cannot communicate with the public network except via the public side of the dual homed gateway. Application level or "proxy" gateways are often used to enhance the functionality of dual homed gateways. Much of the protocol level software on networks operates in a store-and-forward mode. Prior art application level gateways are service-specific store-and-forward programs which commonly operate in user mode instead of at the protocol level.

All of the internetwork gateways known to date suffer from certain disadvantages which compromise their security or inconvenience users. Most known internetwork gateways are also potentially susceptible to intruders if improperly used or configured.

The only firewall for many network installations is a screening router which is positioned between the private network and the public network. The screening router is designed to permit communications only through certain predesignated ports. Many network services are offered on specific designated ports. Generally, screening routers are configured to permit all outbound traffic from the private network while restricting inbound traffic to those certain specific ports allocated to certain network services. A principal weakness of screening routers is that the router's administrative password may be compromised. If an intruder is capable of communicating directly with the router, the intruder can very easily open the entire private network to attack by disabling the screening algorithms. Unfortunately, this is extremely difficult to detect and may go completely unnoted until serious damage has resulted. Screening routers are also subject to permitting vandalism by "piggybacked" protocols which permit intruders to achieve a higher level of access than was intended to be permitted.

Packet filters are a more sophisticated type of screening that operates on the protocol level. Packet filters are generally host-based applications which permit certain communications over predefined ports. Packet filters may have associated rule bases and operate on the principle of "that which is not expressly permitted is prohibited". Public networks such as the Internet operate in TCP/IP protocol. A UNIX operating system running TCP/IP has a capacity of

64K communication ports. It is therefore generally considered impractical to construct and maintain a comprehensive rule base for a packet filter application. Besides, packet filtering is implemented using the simple Internet Protocol (IP) packet filtering mechanisms which are not regarded as being robust enough to permit the implementation of an adequate level of protection. The principal drawback of packet filters is that they are executed by the operating system kernel and there is a limited capacity at that level to perform screening functions. As noted above, protocols may be piggybacked to either bypass or fool packet filtering mechanisms and may permit skilled intruders to access the private network.

The dual homed gateway is an often used and easy to implement alternative. Since the dual homed gateway does not forward TCP/IP traffic, it completely blocks communication between the public and private networks. The ease of use of a dual homed gateway depends upon how it is implemented. It may be implemented by giving users logins to the public side of the gateway host, or by providing application gateways,for specific services. If users are permitted to log on to the gateway, the firewall security is seriously weakened because the risk of an intrusion increases substantially, perhaps exponentially, with each user login due to the fact that logins are a vulnerable part of any security system. Logins are often compromised by a number of known methods and are the usual entry path for intruders.

The alternative implementation of a dual homed gateway is the provision of application gateways for specific network services. Application gateways have recently gained general acceptance as a method of implementing internetwork firewalls. Application gateways provide protection at the application level and the Transmission Control Protocol (TCP) circuit layer. They therefore permit data sensitivity checking and close loopholes left in packet filters. Firewalls equipped with application gateways are commonly labelled application level firewalls. These firewalls operate on the principle of "that which is not expressly permitted is prohibited". Users can only access public services for which an application gateway has been installed on the dual homed gateway. Although application level firewalls are secure, the known firewalls of this type are also inefficient. The principal disadvantage of known application level firewalls is that they are not transparent to the user. They generally require the user to execute time-consuming extra operations or to use specially adapted network service programs. For example, in an open connection to the Internet, a user can Telnet directly to any host on the Internet by issuing the following command:

Telnet target.machine

However if the user is behind an application level firewall, the following command must be issued:

Telnet firewall

After the user has established a connection with the firewall, the user will optionally enter a user ID and a password if the firewall requires authentication. Subsequent to authentication, the user must request that the firewall connect to the final Telnet target machine. This problem is the result of the way in which the UNIX operating system handles IP packets. A standard TCP/IP device will only accept and attempt to process IP packets addressed to itself. Consequently, if a user behind an application firewall issues the command:

Telnet target.machine

an IP packet will be generated by the user workstation that is encapsulated with the device address of the firewall but with an IP destination address of the target.machine. This packet will not be processed by the firewall station and will therefore be discarded because IP packet forwarding has been disabled in the application level firewall.

Known application level firewalls also suffer from the disadvantage that to date application interfaces have been required for each public network service. The known application level firewalls will not support "global service" or applications using "dynamic port allocations" assigned in real time by communicating systems.

Users on private networks having an application level firewall interface therefore frequently install "back doors" to the public network in order to run services for which applications have not been installed, or to avoid the inconvenience of the application gateways. These back doors provide an unscreened, unprotected security hole in the private network which renders that network as vulnerable as if there were no firewall at all.

SUMMARY OF THE INVENTION

It is an object of the invention to provide an internetwork security gateway which overcomes the known disadvantages of prior art internetwork security gateways.

It is a further object of the invention to provide an internetwork security gateway which provides application proxy flexibility, security and control while permitting users to transparently access public network services.

It is a further object of the invention to provide an internetwork security gateway which supports any currently offered or future network service.

It is yet a further object of the invention to provide an internetwork security gateway which supports applications using port numbers that are dynamically assigned in real time by the communicating systems.

It is yet a further object of the invention to provide an internetwork security gateway which listens to all communications ports in order to detect any attempted intrusion into a protected network, regardless of the intruder's point of attack.

In accordance with a first aspect of the invention there is disclosed a method of providing a secure gateway between a private network and a potentially hostile network, comprising the steps of:

a) accepting from either network all communications packets that are encapsulated with a hardware destination address that matches the device address of the gateway;
b) determining whether there is a process bound to a destination port number of an accepted communications packet;
c) establishing a first communications session with a source address/source port of the accepted communications packet if there is a process bound to the destination port number, else dropping the packet;
d) establishing a second communications session with a destination address/destination port number of the accepted communications packet if a first communications session is established; and
e) transparently moving data associated with each subsequent communications packet between the respective first and second communications sessions, whereby the first session communicates with the source and the second session communicates with the destination using the data moved between the first and second sessions.

In accordance with a further aspect of the invention there is disclosed an apparatus for providing a secure gateway for data exchanges between a private network and a potentially hostile network, comprising in combination:

a gateway station adapted for connection to a telecommunications connection with each of the private network and the potentially hostile network;
an operating system executable by the gateway station, a kernel of the operating system having been modified so that the operating system:

a) cannot forward any communications packet from the private network to the potentially hostile network or from the potentially hostile network to the private network; and
b) will accept for processing any communications packet from either of the private network and the potentially hostile network provided that the packet is encapsulated with a hardware destination address that matches the device address of the gateway station on the respective networks; and

at least one proxy process executable by the gateway station, the proxy process being adapted to transparently initiate a first communications session with a source of an initial data packet accepted by the operating system and to transparently initiate a second communications session with a destination of the packet, and to transparently pass a data portion of packets received by the first communications session to the second communications session and to pass the data portion of packets received by the second communications session to the first communications session, whereby the first session communicates with the source using data from the second session and the second session communicates with the destination using data received from the first session.

The invention therefore provides a method and an apparatus which permits a private network to be securely interconnected with a public or a potentially hostile network.

The method in accordance with the invention involves protecting a private network interconnected with a potentially hostile network whereby a gateway between the two networks transparently imitates a host when a communication data packet is received from a client on one of the networks by initiating a communication session with the client. If the client is determined to have access rights to the requested service, the gateway station imitates the client to the host on the other network by initiating a communications session with the host. Thereafter, data is passed between the client session and the host session by a process which coordinates communications between the two distinct, interdependent communications sessions which proceed between the client and the gateway station and the host and the gateway station.

For instance, using a gateway station in accordance with the invention as an internetwork interface, a user on the private network can issue the command:

telnet publictarget.machine

and the command will appear to the user to be executed as if no gateway existed between the networks so long as the user is permitted by the rule bases maintained by the private network security administrator to access the public-target machine.

In order to achieve transparency of operation, the gateway station is modified to accept for processing all IP packets encapsulated in a network operating system capsule (e.g. an ethernet capsule) having a destination address which matches the device address of the gateway station, regardless of the destination address of the IP packet. This modification permits the gateway station to provide transparent service to users on either network, provided the users are authorized for the service. Furthermore, the gateway station in accordance with the invention runs a novel generic

proxy which permits it to listen to all of the 64K communications ports accommodated by the UNIX operating system which are not served by a dedicated proxy process. As is well known to those skilled in the art, certain internetwork services have been assigned specific ports for communication. Most of the designated ports on the Internet are those port numbers in the range of 0-1K (1,024). Other applications and services use port numbers in the range of 1K to 64K. As noted above, the gateway station in accordance with the invention "listens" to all 64K ports. The generic proxy process which is executed by the gateway station responds to any request for service that is not served by a dedicated proxy process, regardless of the destination port number to which the request for service is made. Every request for service may therefore be responded to. When an intruder attacks a private network, the intruder must attempt to access the network through the gateway station. Most firewalls listen to only a limited subset of the available communications ports. An intruder can therefore probe unattended areas of the firewall without detection. The gateway station in accordance with the invention will, however, detect a probe on any port and may be configured to set an alarm condition if repeated probes are attempted. The gateway station in accordance with the invention can also be configured to perform data sensitivity screening because all communications packets are delivered by the kernel to the application level where the data portion of each packet is passed from one in progress communications session to the other. Data sensitivity screening permits the detection of sophisticated intrusion techniques such as piggybacked protocols, and the like.

The apparatus in accordance with the invention is modeled on the concept of a bastion host, preferably configured as a dual home firewall. The apparatus in accordance with the invention may also be configured as a multiple-home firewall, a single-home firewall or a screened subnet. Regardless of the configuration, the apparatus preferably comprises a UNIX station which executes a modified operating system in which IP packet forwarding is disabled. The apparatus in accordance with the invention will not forward any IP packet, process ICMP direct messages nor process any source routing packet between the potentially hostile network and the private network. Without IP packet forwarding, direct communication between the potentially hostile network and the private network are disabled. This is a common arrangement for application level firewalls. The apparatus in accordance with the invention is, however, configured to provide a transparent interface between the interconnected networks so that clients on either network can run standard network service applications transparently without extra procedures, or modifications to accomplish communications across the secure gateway. This maximizes user satisfaction and minimizes the risk of a client establishing a "back door" to a potentially hostile network.

The methods and the apparatus in accordance with the invention therefore provide a novel communications gateway for interconnecting private and public networks which permit users to make maximum use of public services while providing a tool for maintaining an impeccable level of security for the private network.

BRIEF DESCRIPTION OF THE DRAWINGS

A preferred embodiment of the invention will now be further explained by way of example only and with reference to the following drawings, wherein:

FIG. 1 is a schematic diagram of a preferred configuration for an apparatus in accordance with the invention for providing a secure gateway for data exchanges between a private network and a potentially hostile network;

FIG. 2 is a schematic diagram of an IP header, a TCP and a UDP header in accordance with standard TCP/IP format;

FIG. 3 is a schematic diagram of ethernet encapsulation in accordance with RFC 894;

FIG. 4 is a schematic diagram of a communications flow path between a gateway station in accordance with the invention, a client on a private network and a host on a public network;

FIG. 5 is a flow diagram of a general overview of TCP routing by the kernel of a UNIX station in accordance with the prior art;

FIG. 6 is a flow diagram of a general overview of TCP routing by a modified UNIX kernel in accordance with the invention;

FIG. 7a is a first portion of a flow diagram of a general overview of the implementation of the invention at the application level of a gateway station; and

FIG. 7b is a second portion of the flow diagram shown in FIG. 7a.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

Most UNIX hosts communicate using TCP/IP protocol. The preferred embodiment of the invention is therefore constructed from a UNIX station having a UNIX operating system. While the preferred embodiment of the invention described below is explained with particular reference to the UNIX environment, it is to be well understood by those skilled in the art that the principles, concepts and methods described may be readily adapted to function with other

internetwork communication protocols in other operating environments.

FIG. 1 shows a schematic diagram of a private network 10, considered a secure network, connected to a public network 12, considered a potentially hostile network, through a gateway station 14 in accordance with the invention. In this configuration, the gateway station 14 is configured as a dual homed bastion host which provides a secure interface between the private network 10 and the public network 12.

The private network 10 includes a multiplicity of users 16, commonly referred to as clients, and one or more servers such as the name, mail and news server 18. The public network 12 may be, for example, the Internet which comprises hundreds of thousands of interconnected machines. The connection between the private network 10 and the public network 12 passes through a router 20 that relays packets over a communications line 22 which may be an ISDN, 56K level BPS, T1 or a dial up connection depending on the private network's arrangement with a public network service provider. All of the private network 10 on the private side of the gateway station 14 is unavailable to the public and is not advertised on the public network 12. The gateway station 14 does not broadcast any routes to the public network 12 and the gateway router 20 should not have any static routes defined for the private network 10. The private network 10 is therefore "routing invisible" to the public network 12. The hosts in the private network 10 can use private network addresses as defined in RFC-1597.

As can be seen in FIG. 1, the gateway station has one device address for its communications connection with the private network 10 and another device address for its communications connection with the public network 12. The device address on the public side of the gateway station 14 must be advertised on the public network 12. The private network 10 may also include one or more public servers 24 which may be accessed directly from the public network 12. The public servers 24 must have addresses which are registered and published on the public network 12. The private network 10 can only access the public servers 24 through the gateway station 14. The public server 24 is treated like any other server on the public network 12 when a user 16 initiates communication from the private network 10.

The gateway station 14 is preferably a UNIX station, well known in the art. The kernel of the operating system of the gateway station 14 is modified to disable all IP forwarding, source routing and IP redirecting functions. It is therefore impossible to have communication data packets flow directly through the gateway station 14. As will be explained below in some detail, these functions have been replaced with processes which ensure that all communications data packets from the private network 10 to the public network 12, or vice versa, are properly authenticated.

Public network communications are typically in TCP/IP format. FIG. 2 shows a schematic diagram of an IP header 26, a TCP header 28 and a UDP (User Datagram Protocol) header 30. Each IP header includes a 32-bit source IP address 32 and a 32-bit destination IP address 34. Each TCP header and each UDP header include a 16-bit source port number 36 and a 16-bit destination port number 38. Each communication data packet therefore includes a source address/source port number and a destination address/destination port number, in accordance with this communications protocol which is well known in the art. In addition to the TCP/IP communications protocol, local area networks often operate using ethernet network control software which handles intranetwork communications. In accordance with ethernet protocol, TCP/IP packets are encapsulated with an ethernet encapsulation packet to facilitate routing and ensure error free transmission.

FIG. 3 shows a schematic diagram of an ethernet encapsulation packet in accordance with RFC 894. Each encapsulation includes an ethernet destination address 40, an ethernet source address 42 and a check sum 44 for facilitating error detection and correction.

FIG. 4 illustrates schematically a typical communications session between a client station 16 on the private network 10 and a public host 46 on the public network 12. All communications between the networks are handled by the gateway station 14. When a client 16 wishes to communicate with the public network 12, such as in accessing a public host 46, the client 16 issues a network command as if the client were not behind a firewall. For instance, client 16 may issue the command:

Telnet Target.Machine

The private network 10 is configured so that all packets directed to the public network 12 are encapsulated with the ethernet destination address (192.168.77.1) of the gateway station 14. A TCP/LP packet encapsulated with the ethernet destination address of the gateway station 14 is therefore dispatched by the client 16. A normally configured UNIX device will not accept for processing TCP/IP packets which do not have an IP destination address equal to its own IP address. The kernel of the operating system of the gateway station 14 is modified so that the gateway station 14 will accept for processing any TCP/IP packet having an encapsulation destination address 40 that matches the device address of the gateway station 14. When the gateway station 14 receives the client packet containing the Telnet command, a process is initiated on the gateway station 14 which responds to the client 16 to establish a communication session 17 as if it were the target machine. As will be explained below in detail, the process then authenticates the client's authorization to access the requested service and if the client 16 is determined to have the required authorization, the gateway station 14 initiates a second communications process 19 with the remote host 46 in which the gateway station 14 simulates the client 16 without revealing the client address. Once the two communication sessions 17, 19 are operative, communication is effected between the client 16 and the host 46 by passing communication data

between the two interdependent communication sessions. This is accomplished by a process that operates at the application level on the gateway station 14, as will be explained in detail below. The process accepts communication packets passed from the IP layer of the gateway station 14 by the modified operating system kernel, extracts the data from the packets and passes the data to the appropriate interdependent communications session. Data sensitivity checking is preferably performed before packet data is released to the appropriate interdependent communications session. Data sensitivity checking can prevent protocol piggybacking and other sophisticated intrusion techniques. Data sensitivity checking is performed using application level algorithms, some of which are well known in the art, but new algorithms are still being developed. All of this processing is completely transparent to client 16 because all communications appear to be direct to host 46. Host 46 likewise cannot detect that communications with client 16 are not direct.

FIG. 5 shows a general overview of the way in which a prior art UNIX operating system kernel handles data communication packets. In a first step 48, a data packet is received by a UNIX workstation (not illustrated). The encapsulation address (typically an ethernet encapsulation destination address 40, see FIG. 3) is checked to determine whether it matches a device address of the station in step 50. If the addresses do not match, the packet is dropped in step 52. If the addresses match, the IP destination address 34 (see FIG. 2) is examined to determine whether it matches an IP address of the station in step 54. If a match is found, the destination port is examined in step 56 to determine whether there is a process bound to a communications port indicated as the destination port by a TCP or UDP portion of the packet that indicates the destination port number 38 (see FIG. 2). If no process is bound to the destination port, the packet is dropped in step 58. Otherwise, the kernel starts a TCP or a UDP session with the IP source in step 59, and delivers the packet to the bound process in step 60. Thereafter, the packet is processed by the process bound to the destination port in step 61, in a manner well known in the art. If the IP destination address does not match any IP address for the station, the kernel attempts to forward the packet in step 62 by consulting routing tables in a process which is also well understood by those skilled in the art.

FIG. 6 is a flow diagram of a general overview of packet processing by a UNIX operating system kernel modified in accordance with the invention. In order to understand the process completely, it is important to understand that the gateway station 14 (see FIG. 1) is configured by a systems administrator using configuration programs supplied with the gateway station 14. When the gateway station 14 is initialized, a system configuration file is examined to determine what network services are to be supported by the gateway station 14. In order to maximize performance efficiency of the gateway station 14, commonly used services are supported by processes adapted to most efficiently handle communications for each respective service. These processes are called "proxies". On system initialization, any proxy given operating rights by the system administrator is said to "bind" to the port to which the proxy has been assigned. Thereafter, the process is said to be "bound" to the port. In accordance with the invention, the gateway station 14 is also supplied with a generic proxy that is assigned to port 59813. This port assignment is an arbitrary assignment and another port number may be used. When the gateway station 14 is initialized, the generic proxy binds to port 59813, provided that the systems administrator has given it operating rights to do so.

With reference again to FIG. 6, the modified kernel receives a packet in step 64 in the same manner as an unmodified prior art kernel. The packet is examined to determine whether the encapsulation destination address 40 (see FIG. 3) equals the device address of the gateway station 14 in step 66. If no match is found, the packet is dropped in step 68. If the encapsulation destination address 40 equals the device address of the gateway station 14, the destination port of the TCP or UDP portion of the packet is examined to determine the destination port number 38 (see FIG. 2) and whether there is a process bound to that destination port in step 70. The IP destination address 34 of the packet is ignored while making this determination. If no proxy process is bound to the destination port of the data packet, the kernel checks port 59813 to determine whether the generic proxy process is bound to that port in step 72. If the generic proxy process is not bound to port 59813, the packet is dropped in step 74. If it is determined that a proxy process is bound to a port which can serve the destination port number 38 in either of steps 70 or 72, a session (TCP or UDP) is initiated with the packet source IP address 32 in step 76 and in step 78, the packet is delivered by the kernel to the proxy process designated in steps 70, 72. This critical modification of the operating system kernel permits the kernel, within the permission boundaries imposed by a systems administrator, to "listen" to all 64K ports available for communication. Appendix A attached hereto is a printed listing of the modified source code in the kernel.

FIGs. 7a and 7b show a flow diagram of a general overview of processing at the application or proxy level of operations on the gateway station 14 in accordance with the invention. While this general overview shows a simple single user process, it will be understood by those skilled in the art that all proxies in accordance with the invention are multi-tasking proxies which can service a plurality of users simultaneously by creating "sockets" for multi-task organization in a manner well understood in the art.

In step 80 of FIG. 7a, the proxy is initialized. The proxy may be any one of a number of proxies. As noted above, in order to maximize the performance of a gateway station 14, certain customized proxies handle certain services. In particular, the preferred embodiment of the invention includes at least the following customized proxies:

proxy - Telnet (port 23)
proxy - FTP (port 21)
proxy - Gopher (port 70)
proxy - TCP (port xxx or port 59813)
UDP - relay (port xxx or port 59813)

These custom proxies are adapted to most efficiently handle the services with which they are associated. Other custom proxies can, of course, be added to the proxy processes which are bound to ports on the gateway station 14. If a data packet has a destination port number 38 that is determined to point to a port to which a custom proxy process is bound, the packet is passed by the kernel to that custom proxy process in step 82 where the custom proxy process waits for packets to arrive on the port to which it is bound, represented by "port xxx" in the diagram. As explained above with relation to FIG. 6, if the destination port number 38 of a data packet does not point to a port to which a custom proxy process is bound, the packet is delivered by the kernel to port 59813. Proxy 59813 is the generic proxy process which is designated to handle any request for service for which a customized proxy process does not exist. Although the generic proxy process is literally bound to a specific port, such as port 59813, in combination with the modified kernel it is operationally bound to every port to which a custom proxy process is not bound. Thus a versatile gateway station 14 which can handle any TCP/IP communications session is provided.

When the packet is passed by the kernel in step 82 to a bound proxy process, the proxy process determines whether the source IP address 32 is permitted to communicate with the destination IP address in step 84. This is accomplished by reference to a rule base maintained by a systems administrator of the private network. Rule bases are well understood in the art and their structure is common general knowledge. Preferably, the rule base in accordance with the preferred embodiment of the invention includes a minimum of the following elements in the rule set to determine authorization:

IP source address
IP destination address
Service required (Telnet, Archie, Gopher, WAIS, etc.)
User ID
Password

If the IP source address 32 is determined in step 84 not to be authorized for the requested service, the gateway station 14 drops the communication session in step 86. Otherwise, the proxy determines whether user level authentication is required in step 88. User level authentication is also under the control of the systems administrator. For example, the systems administrator may not require any authentication of users on the private network on the theory that the private network is only accessible to secure individuals. On the other hand, the systems administrator has the option of requiring user level authentication for all users or any selected user. Even if user level authentication is required, in accordance with the preferred embodiment of the invention a user can authenticate and enable transparent mode wherein authentication need not be repeated for subsequent sessions for as long as authentication is maintained, as will be explained below in more detail.

If user level authentication is determined to be required in step 88, the proxy process authenticates the user in step 90. Authentication is preferably accomplished by requiring the user to enter an identification code and a password in a manner well known in the art. The identification code and the password entered are verified in a user authentication data base (not illustrated), the structure and operation of which are also well known in the art. In step 92, the gateway station 14 determines whether the user has authenticated by referencing the user authentication data base. If the user is not successfully authenticated, the session is dropped in step 94. If it is determined that no authentication is required in step 88, or if the user successfully authenticates in step 92, the IP destination address 34 of the packet is examined to determine if it is an IP address of the gateway station 14 in step 96 (see FIG. 7b). Step 96 is preferably only executed by the custom proxies for Telnet, FTP and Gopher, as will be explained below in detail. If the IP destination address 34 of the packet corresponds to an IP address of the gateway station 14, the gateway station starts a session to permit the user to enable or disable transparent mode in step 98, as will also be explained below. In step 100, the proxy process waits for communication packets to arrive on the port "xxx", which is the port on which the communications session was initiated, where it is delivered by the modified kernel. When a communications packet arrives, it is processed by the proxy process to permit the IP source to enable or disable transparent mode in step 101. Thereafter, the proxy process determines in step 102 whether the session has ended and if not it returns to step 100 to wait for a data packet to arrive on the port. Otherwise it returns to step 80 (see FIG. 7a) to initialize for a new session. If the IP destination address 34 of the packet does not match an IP address of the gateway station 14, a second communication session is established in step 104. The second communication session is established between the gateway station 14 and the IP destination address 34/TCP destination port number 38 indicated by the packet. The second communication

session established in step 104 is established exclusively by the proxy process, all operations being completely transparent to the IP source. Once the second communications session is established, the proxy process waits for data packets from each session in step 106. The data packets are delivered by the kernel to the proxy process which relays the data portion of each packet from one session to the other so that the two interdependent sessions appear to the IP source and to the IP destination to be one direct session between a client source and a host destination. Data sensitivity checking may be accomplished by the proxy process before relaying the data from one session to the other. Data sensitivity checks ensure that the gateway station 14 provides a very secure interface that is, for all practical purposes, impossible for intruders to breach.

Thus a completely transparent firewall is provided which permits a security administrator to exercise potent control over the access to the private network. In step 108, the proxy process checks to determine whether either of the sessions is terminated. If either session terminates, the other session is likewise terminated and the proxy process returns to step 80 for initialization. The custom proxy processes in accordance with the invention are public domain proxy processes which have been modified to cooperate with the kernel in accordance with the methods of the invention. Appendix B attached hereto is a printed listing of the modifications made to the public domain proxies for the Telnet proxy and the FTP proxy. The principles illustrated may be applied by a person skilled in the art to any public domain or proprietary proxy process.

One of the significant features of the invention is the fact that the proxy process which executes in accordance with the flow diagram shown in FIG. 7 may be the generic proxy process, which can handle any request for service for which a customized proxy process does not exist. The generic proxy process in accordance with the invention is a proprietary TCP proxy. A complete source code listing of the proxy TCP is attached hereto as Appendix D. As explained above, in the preferred embodiment of the invention, the generic proxy process is bound to port 59813, but the specific port to which the proxy process is bound is substantially immaterial. The generic proxy process increases many fold the versatility of gateway station 14 and permits the gateway station to handle any communications session, including sessions which use port numbers dynamically allocated in real time by the communicating systems. It also provides a secure gateway which is adapted to support, without modification, new service offerings on the public network 12. If a new service offering becomes a popular service for which there is demand, a customized proxy can be written for that service. During development and testing of the customized proxy, however, the new service can be supported by the generic proxy process. Thus a versatile, secure internetwork gateway is provided which supports any known or future service available on a public network.

The gateway is completely transparent to users in all instances except when a user authentication is requested. In that instance, users may be authorized to enter authentication requests to enable a transparent mode of operation wherein subsequent sessions require no further authentication, and the gateway station 14 is completely transparent to the user. In accordance with the preferred embodiment of the invention, transparent mode can be enabled using a Telnet session, an FTP session or a Gopher session. To enable transparent mode using the Telnet program, a Telnet session is started with gateway station 14 as follows:

your-host% **telnet gatewaystation.company.com**
Trying 198.53.64.2
Connected to gatewaystation.company.com
Escape character is '^]'
gatewaystation proxy-telnet ready:
Username: **You**
Password: **xxxxxxx**
Login Accepted
proxy-telnet> **enable**
proxy-telnet> **quit**
Disconnecting.....
Connection closed by foreign host.
your-host%

A user may also authenticate and enable transparent mode using the FTP program as follows:

your-host% **FTP gatewaystation.company.com**
Connected to gatewaystation.company.com
220 gatewaystation proxy-FTP ready:
Name (gatewaystation.company.com:you): **You**
331 Enter authentication password for you
Password: **xxxxxxx**

```
230 User authentication to proxy
ftp > quote enable
Transparent mode enabled
ftp > quit
your-host%
```

In the preferred embodiment of the invention, a proprietary Gopher proxy is enabled to automatically initiate transparent mode after the user has successfully authenticated to the gateway station 14 by entering a valid user identification and password, whenever a Gopher session is requested and user authentication is required. This user authentication capability is a novel feature for a Gopher proxy. The proprietary source code for the novel Gopher proxy is appended hereto as Appendix C.

The modes for implementing transparent mode are, of course, arbitrary and may be redesigned or reassigned to other programs or proxies as those skilled in the art deem appropriate. Once the transparent mode is enabled, an authentication directory is updated by creating a file entry for the source IP address 32. The authentication files include a creation time variable which is automatically set to the system time when the file is created. This creation time variable is used to track the time of authentication. The files also include a last modification time variable which is automatically updated by the system each time the file is modified. By rewriting the authentication file each time a user initiates a new communications session through the gateway station 14 the time of last use of the gateway station can be tracked. This authentication directory is inspected periodically and user files are deleted from the authentication directory based on any number of predetermined criteria. In accordance with the preferred embodiment, the user file is deleted from the authentication directory if the user has not initiated a communications session through the gateway station for a period of time predefined by the systems administrator. In addition, the user file may be deleted from the authentication directory at a predetermined time of day defined by the system administrator. It is therefore possible to have the authentication of all users of the gateway station 14 revoked at a specified time of day, such as the end of the business day. This further fortifies the security of the gateway.

It is apparent that a novel and particularly invulnerable gateway has been invented. The gateway is efficient as well as secure. It will be readily apparent to those skilled in the art that modification may be made to the preferred embodiment described above without departing from the scope of the invention as expressed in the appended claims.

APPENDIX A

MODIFIED UNIX KERNEL SOURCE CODE LISTING

```
/*
 * Copyright (c) 1982, 1986, 1988, 1993
 *      Regents of the University of California.  All rights reserved.
 *
 * Red   ribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 * 3. All advertising materials mentioning features or use of this software
 *    must display the following acknowledgement:
 *      This product includes software developed by the University of
 *      California, Berkeley and its contributors.
 * 4. Neither the name of the University nor the names of its contributors
 *    may be used to endorse or promote products derived from this software
 *    without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS ``AS IS'' AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 *
 *      @(#)ip_input.c  7.19 (Berkeley) 5/25/91
 */

#include "param.h"
#include "systm.h"
#include "malloc.h"
#include "mbuf.h"
#include "domain.h"
#include "protosw.h"
#include "socket.h"
#include "errno.h"
#include "time.h"
#include "kernel.h"

#include "../net/if.h"
#include "../net/route.h"

#include "in.h"
#include "in_systm.h"
#include "ip.h"
#include "in_pcb.h"
#include "in_var.h"
#include "ip_var.h"
#include "ip_icmp.h"

#if defined(MILKYWAY_BH) || defined(MILKYWAY_VPN)
#include "milkyway/vpn_data.h"
#define GATEWAY
#endif

/* the following are in in_proto.c for easy config-dependence */
extern  int ipforwarding;       /* act as router? */
extern  int ipforward_srcrt;    /* forward src-routed packets? */
extern  int ipsendredirects;
```

```
#ifdef GWSCREEN
extern  void (*ip_forward_fn) __P((struct mbuf *, int));
#endif
void ip_forward __P((struct mbuf *, int));

#ifdef DEBUG
int     ipprintfs = 1;
#endif

extern  struct domain inetdomain;
extern  struct protosw inetsw[];
u_char  ip_protox[IPPROTO_MAX];
int     ipqmaxlen = IFQ_MAXLEN;
struct  in_ifaddr *in_ifaddr;                   /* first inet address */

/*
 * We need to save the IP options in case a protocol wants to respond
 * to an incoming packet over the same route if the packet got here
 * using IP source routing.  This allows connection establishment and
 * maintenance when the remote end is on a network that is not known
 * to us.
 */
int     ip_nhops = 0;
static  struct ip_srcrt {
        struct  in_addr dst;                    /* final destination */
        char    nop;                            /* one NOP to align */
        char    srcopt[IPOPT_OFFSET + 1];       /* OPTVAL, OLEN and OFFSET */
        struct  in_addr route[MAX_IPOPTLEN/sizeof(struct in_addr)];
} ip_srcrt;

#ifdef GATEWAY
extern  int if_index;
u_long  *ip_ifmatrix;
#endif

/*
 * IP initialization: fill in IP protocol switch table.
 * All protocols not implemented in kernel go to raw IP protocol handler.
 */
ip_init()
{
        register struct protosw *pr;
        register int i;

        pr = pffindproto(PF_INET, IPPROTO_RAW, SOCK_RAW);
        if (pr == 0)
                panic("ip_init");
        for (i = 0; i < IPPROTO_MAX; i++)
                ip_protox[i] = pr - inetsw;
        for (pr = inetdomain.dom_protosw;
            pr < inetdomain.dom_protoswNPROTOSW; pr++)
                if (pr->pr_domain->dom_family == PF_INET &&
                    pr->pr_protocol && pr->pr_protocol != IPPROTO_RAW)
                        ip_protox[pr->pr_protocol] = pr - inetsw;
        ipq.next = ipq.prev = &ipq;
        ip_id = time.tv_sec & 0xffff;
        ipintrq.ifq_maxlen = ipqmaxlen;
#ifdef GATEWAY
        i = (if_index + 1) * (if_index + 1) * sizeof (u_long);
        if ((ip_ifmatrix = (u_long *) malloc(i, M_RTABLE, M_WAITOK)) == 0)
                panic("no memory for ip_ifmatrix");
#endif
}

struct  ip *ip_reass();
struct  sockaddr_in ipaddr = { sizeof(ipaddr), AF_INET };
```

```
struct  route ipforward_rt;

/*
 * Ip input routine.  Checksum and byte swap header.  If fragmented
 * try to reassemble.  Process options.  Pass to next level.
 */
ipintr()
{
        register struct ip *ip;
        register struct mbuf *m;
        register struct ipq *fp;
        register struct in_ifaddr *ia;
        int hlen, s;
        struct ifnet *ifp;

next:
        /*
         * Get next datagram off input queue and get IP header
         * in first mbuf.
         */
        s = splimp();
        IF_DEQUEUE(&ipintrq,m);
        splx(s);
        if (m == 0)
                return;

#ifdef  DIAGNOSTIC
        if ((m->m_flags & M_PKTHDR) == 0)
                panic("ipintr no HDR");
#endif
        ifp=m->m_pkthdr.rcvif;

        /*
         * If no IP addresses have been set yet but the interfaces
         * are receiving, can't do anything with incoming packets yet.
         */
        if (in_ifaddr == NULL)
                goto bad;
        ipstat.ips_total++;
        if (m->m_len < sizeof (struct ip) &&
            (m = m_pullup(m, sizeof (struct ip))) == 0) {
                ipstat.ips_toosmall++;
                goto next;
        }
        ip = mtod(m, struct ip *);

        if (ip->ip_v != IPVERSION) {
                /*ipstat.ips_badver++;*/
                goto bad;
        }

        hlen = ip->ip_hl << 2;
        if (hlen < sizeof(struct ip)) { /* minimum header length */
                ipstat.ips_badhlen++;
                goto bad;
        }
        if (hlen > m->m_len) {
                if ((m = m_pullup(m, hlen)) == 0) {
                        ipstat.ips_badhlen++;
                        goto next;
                }
                ip = mtod(m, struct ip *);
        }
        if (ip->ip_sum = in_cksum(m, hlen)) {
                ipstat.ips_badsum++;
                goto bad;
```

```
        }

        /*
         * Convert fields to host representation.
         */
        NTOHS(ip->ip_len);
        if (ip->ip_len < hlen) {
                ipstat.ips_badlen++;
                goto bad;
        }
        NTOHS(ip->ip_id);
        NTOHS(ip->ip_off);

        /*
         * Check that the amount of data in the buffers
         * is as at least much as the IP header would have us expect.
         * Trim mbufs if longer than we expect.
         * Drop packet if shorter than we expect.
         */
        if (m->m_pkthdr.len < ip->ip_len) {
                ipstat.ips_tooshort++;
                goto bad;
        }
        if (m->m_pkthdr.len > ip->ip_len) {
                if (m->m_len == m->m_pkthdr.len) {
                        m->m_len = ip->ip_len;
                        m->m_pkthdr.len = ip->ip_len;
                } else
                        m_adj(m, ip->ip_len - m->m_pkthdr.len);
        }

        /*
         * Process options and, if not destined for us,
         * ship it on.  ip_dooptions returns 1 when an
         * error was detected (causing an icmp message
         * to be sent and the original packet to be freed).
         */
        ip_nhops = 0;           /* for source routed packets */
        if (hlen > sizeof (struct ip) && ip_dooptions(m))
                goto next;

        /*
         * Check our list of addresses, to see if the packet is for us.
         */
        for (ia = in_ifaddr; ia; ia = ia->ia_next) {
#define satosin(sa)     ((struct sockaddr_in *)(sa))

                if (IA_SIN(ia)->sin_addr.s_addr == ip->ip_dst.s_addr)
                        goto ours;
                if (
#ifdef  DIRECTED_BROADCAST
                    ia->ia_ifp == m->m_pkthdr.rcvif &&
#endif
                    (ia->ia_ifp->if_flags & IFF_BROADCAST)) {
                        u_long t;

                        if (satosin(&ia->ia_broadaddr)->sin_addr.s_addr ==
                            ip->ip_dst.s_addr)
                                goto bcast;
                        if (ip->ip_dst.s_addr == ia->ia_netbroadcast.s_addr)
                                goto bcast;
                        /*
                         * Look for all-0's host part (old broadcast addr),
                         * either for subnet or net.
                         */
                        t = ntohl(ip->ip_dst.s_addr);
```

```
                        if (t == ia->ia_subnet)
                                goto bcast;
                        if (t == ia->ia_net)
                                goto bcast;
                }
        }
#ifdef MULTICAST
        if (IN_MULTICAST(ntohl(ip->ip_dst.s_addr))) {
                struct in_multi *inm;
/* #ifdef MROUTING */
                extern struct socket *ip_mrouter;

                if (ip_mrouter) {
                        /*
                         * If we are acting as a multicast router, all
                         * incoming multicast packets are passed to the
                         * kernel-level multicast forwarding function.
                         * The packet is returned (relatively) intact; if
                         * ip_mforward() returns a non-zero value, the packet
                         * must be discarded, else it may be accepted below.
                         *
                         * (The IP ident field is put in the same byte order
                         * as expected when ip_mforward() is called from
                         * ip_output().)
                         */
                        ip->ip_id = htons(ip->ip_id);
                        if (ip_mforward(ip, m->m_pkthdr.rcvif, m) != 0) {
                                m_freem(m);
                                goto next;
                        }
                        ip->ip_id = ntohs(ip->ip_id);

                        /*
                         * The process-level routing daemon needs to receive
                         * all multicast IGMP packets, whether or not this
                         * host belongs to their destination groups.
                         */
                        if (ip->ip_p == IPPROTO_IGMP)
                                goto ours;
                }
/* #endif */
                /*
                 * See if we belong to the destination multicast group on the
                 * arrival interface.
                 */
                IN_LOOKUP_MULTI(ip->ip_dst, m->m_pkthdr.rcvif, inm);
                if (inm == NULL) {
                        m_freem(m);
                        goto next;
                }
                goto ours;
        }
#endif
        if (ip->ip_dst.s_addr == (u_long)INADDR_BROADCAST)
                goto bcast;
        if (ip->ip_dst.s_addr == INADDR_ANY)
                goto ours;      /* bcast? */

#if defined(MILKYWAY_BH)
 /* mcr - 94/10/11
    Do not forward and simply give the packet to the upper layer
    Just to see how it goes
  */

#if defined(MILKYWAY_VPN)
        /* but we need to be a bit smarter for VPN */
```

```
        if(vpn_forward(m)) {
          goto next;
        } else {
          goto ours;

#endif

#else
        /*
         * Not for us; forward if possible and desirable.
         */
        if (ipforwarding == 0) {
                ipstat.ips_cantforward++;
                m_freem(m);
        } else
#ifdef GWSCREEN
                (*ip_forward_fn)(m, 0);
#else
                ip_forward(m, 0);
#endif
        goto next;
#endif /* MILKYWAY_BH */

bcast:
        /* set the BCAST bit on packets destined to an IP broadcast address */
        m->m_flags |= M_BCAST;

ours:
        /*
         * If offset or IP_MF are set, must reassemble.
         * Otherwise, nothing need be done.
         * (We could look in the reassembly queue to see
         * if the packet was previously fragmented,
         * but it's not worth the time; just let them time out.)
         */
        if (ip->ip_off &~ IP_DF) {
                if (m->m_flags & M_EXT) {               /* XXX */
                        if ((m = m_pullup(m, sizeof (struct ip))) == 0) {
                                ipstat.ips_toosmall++;
                                goto next;
                        }
                        ip = mtod(m, struct ip *);
                }
                /*
                 * Look for queue of fragments
                 * of this datagram.
                 */
                for (fp = ipq.next; fp != &ipq; fp = fp->next)
                        if (ip->ip_id == fp->ipq_id &&
                            ip->ip_src.s_addr == fp->ipq_src.s_addr &&
                            ip->ip_dst.s_addr == fp->ipq_dst.s_addr &&
                            ip->ip_p == fp->ipq_p)
                                goto found;
                fp = 0;
found:

                /*
                 * Adjust ip_len to not reflect header,
                 * set ip_mff if more fragments are expected,
                 * convert offset of this to bytes.
                 */
                ip->ip_len -= hlen;
                ((struct ipasfrag *)ip)->ipf_mff = 0;
                if (ip->ip_off & IP_MF)
                        ((struct ipasfrag *)ip)->ipf_mff = 1;
                ip->ip_off <<= 3;
```

```
                /*
                 * If datagram marked as having more fragments
                 * or if this is not the first fragment,
                 * attempt reassembly; if it succeeds, proceed.
                 */
                if (((struct ipasfrag *)ip)->ipf_mff || ip->ip_off) {
                        ipstat.ips_fragments++;
                        ip = ip_reass((struct ipasfrag *)ip, fp);
                        if (ip == 0)
                                goto next;
                        else
                                ipstat.ips_reassembled++;
                        m = dtom(ip);
                } else
                        if (fp)
                                ip_freef(fp);
        } else
                ip->ip_len -= hlen;

        /*
         * Switch out to protocol's input routine.
         */
        ipstat.ips_delivered++;
        (*inetsw[ip_protox[ip->ip_p]].pr_input)(m, hlen);
        goto next;
bad:
        m_freem(m);
        goto next;
}

/*
 * Take incoming datagram fragment and try to
 * reassemble it into whole datagram.  If a chain for
 * reassembly of this datagram already exists, then it
 * is given as fp; otherwise have to make a chain.
 */
struct ip *
ip_reass(ip, fp)
        register struct ipasfrag *ip;
        register struct ipq *fp;
{
        register struct mbuf *m = dtom(ip);
        register struct ipasfrag *q;
        struct mbuf *t;
        int hlen = ip->ip_hl << 2;
        int i, next;

        /*
         * Presence of header sizes in mbufs
         * would confuse code below.
         */
        m->m_data += hlen;
        m->m_len -= hlen;

        /*
         * If first fragment to arrive, create a reassembly queue.
         */
        if (fp == 0) {
                if ((t = m_get(M_DONTWAIT, MT_FTABLE)) == NULL)
                        goto dropfrag;
                fp = mtod(t, struct ipq *);
                insque(fp, &ipq);
                fp->ipq_ttl = IPFRAGTTL;
                fp->ipq_p = ip->ip_p;
                fp->ipq_id = ip->ip_id;
```

```
                fp->ipq_next = fp->ipq_prev = (struct ipasfrag *)fp;
                fp->ipq_src = ((struct ip *)ip)->ip_src;
                fp->ipq_dst = ((struct ip *)ip)->ip_dst;
                q = (struct ipasfrag *)fp;
                goto insert;
        }

        /*
         * Find a segment which begins after this one does.
         */
        for (q = fp->ipq_next; q != (struct ipasfrag *)fp; q = q->ipf_next)
                if (q->ip_off > ip->ip_off)
                        break;

        /*
         * If there is a preceding segment, it may provide some of
         * our data already.  If so, drop the data from the incoming
         * segment.  If it provides all of our data, drop us.
         */
        if (q->ipf_prev != (struct ipasfrag *)fp) {
                i = q->ipf_prev->ip_off + q->ipf_prev->ip_len - ip->ip_off;
                if (i > 0) {
                        if (i >= ip->ip_len)
                                goto dropfrag;
                        m_adj(dtom(ip), i);
                        ip->ip_off += i;
                        ip->ip_len -= i;
                }
        }

        /*
         * While we overlap succeeding segments trim them or,
         * if they are completely covered, dequeue them.
         */
        while (q != (struct ipasfrag *)fp && ip->ip_off + ip->ip_len > q->ip_off
) {
                i = (ip->ip_off + ip->ip_len) - q->ip_off;
                if (i < q->ip_len) {
                        q->ip_len -= i;
                        q->ip_off += i;
                        m_adj(dtom(q), i);
                        break;
                }
                q = q->ipf_next;
                m_freem(dtom(q->ipf_prev));
                ip_deq(q->ipf_prev);
        }

insert:
        /*
         * Stick new segment in its place;
         * check for complete reassembly.
         */
        ip_enq(ip, q->ipf_prev);
        next = 0;
        for (q = fp->ipq_next; q != (struct ipasfrag *)fp; q = q->ipf_next) {
                if (q->ip_off != next)
                        return (0);
                next += q->ip_len;
        }
        if (q->ipf_prev->ipf_mff)
                return (0);

        /*
         * Reassembly is complete; concatenate fragments.
         */
```

```
                q = fp->ipq_next;
                m = dtom(q);
                t = m->m_next;
                m->m_next = 0;
                 _cat(m, t);
                q = q->ipf_next;
                while (q != (struct ipasfrag *)fp) {
                        t = dtom(q);
                        q = q->ipf_next;
                        m_cat(m, t);
                }

                /*
                 * Create header for new ip packet by
                 * modifying header of first packet;
                 * dequeue and discard fragment reassembly header.
                 * Make header visible.
                 */
                ip = fp->ipq_next;
                ip->ip_len = next;
                ((struct ip *)ip)->ip_src = fp->ipq_src;
                ((struct ip *)ip)->ip_dst = fp->ipq_dst;
                remque(fp);
                (void) m_free(dtom(fp));
                m = dtom(ip);
                m->m_len += (ip->ip_hl << 2);
                m->m_data -= (ip->ip_hl << 2);
                /* some debugging cruft by sklower, below, will go away soon */
                if (m->m_flags & M_PKTHDR) { /* XXX this should be done elsewhere */
                        register int plen = 0;
                        for (t = m; m; m = m->m_next)
                                plen += m->m_len;
                        t->m_pkthdr.len = plen;
                }
                return ((struct ip *)ip);

        dropfrag:
                ipstat.ips_fragdropped++;
                m_freem(m);
                return (0);
        }

        /*
         * Free a fragment reassembly header and all
         * associated datagrams.
         */
        ip_freef(fp)
                struct ipq *fp;
        {
                register struct ipasfrag *q, *p;

                for (q = fp->ipq_next; q != (struct ipasfrag *)fp; q = p) {
                        p = q->ipf_next;
                        ip_deq(q);
                        m_freem(dtom(q));
                }
                remque(fp);
                (void) m_free(dtom(fp));
        }

        /*
         * Put an ip fragment on a reassembly chain.
         * Like insque, but pointers in middle of structure.
         */
        ip_enq(p, prev)
                register struct ipasfrag *p, *prev;
```

```
{

        p->ipf_prev = prev;
        p->ipf_next = prev->ipf_next;
        prev->ipf_next->ipf_prev = p;
        prev->ipf_next = p;
}

/*
 * To ip_enq as remque is to insque.
 */
ip_deq(p)
        register struct ipasfrag *p;
{

        p->ipf_prev->ipf_next = p->ipf_next;
        p->ipf_next->ipf_prev = p->ipf_prev;
}

/*
 * IP timer processing;
 * if a timer expires on a reassembly
 * queue, discard it.
 */
ip_slowtimo()
{
        register struct ipq *fp;
        int s = splnet();

        fp = ipq.next;
        if (fp == 0) {
                splx(s);
                return;
        }
        while (fp != &ipq) {
                --fp->ipq_ttl;
                fp = fp->next;
                if (fp->prev->ipq_ttl == 0) {
                        ipstat.ips_fragtimeout++;
                        ip_freef(fp->prev);
                }
        }
        splx(s);
}

/*
 * Drain off all datagram fragments.
 */
ip_drain()
{

        while (ipq.next != &ipq) {
                ipstat.ips_fragdropped++;
                ip_freef(ipq.next);
        }
}

extern struct in_ifaddr *ifptoia();
struct in_ifaddr *ip_rtaddr();

/*
 * Do option processing on a datagram,
 * possibly discarding it if bad options are encountered,
 * or forwarding it if source-routed.
 * Returns 1 if packet has been forwarded/freed,
 * 0 if the packet should be processed further.
```

```
 */
ip_dooptions(m)
        struct mbuf *m;
{
        register struct ip *ip = mtod(m, struct ip *);
        register u_char *cp;
        register struct ip_timestamp *ipt;
        register struct in_ifaddr *ia;
        int opt, optlen, cnt, off, code, type = ICMP_PARAMPROB, forward = 0;
        struct in_addr *sin;
        n_time ntime;

        cp = (u_char *)(ip + 1);
        cnt = (ip->ip_hl << 2) - sizeof (struct ip);
        for (; cnt > 0; cnt -= optlen, cp += optlen) {
                opt = cp[IPOPT_OPTVAL];
                if (opt == IPOPT_EOL)
                        break;
                if (opt == IPOPT_NOP)
                        optlen = 1;
                else {
                        optlen = cp[IPOPT_OLEN];
                        if (optlen <= 0 || optlen > cnt) {
                                code = &cp[IPOPT_OLEN] - (u_char *)ip;
                                goto bad;
                        }
                }
                switch (opt) {

                default:
                        break;

                /*
                 * Source routing with record.
                 * Find interface with current destination address.
                 * If none on this machine then drop if strictly routed,
                 * or do nothing if loosely routed.
                 * Record interface address and bring up next address
                 * component.  If strictly routed make sure next
                 * address is on directly accessible net.
                 */
                case IPOPT_LSRR:
                case IPOPT_SSRR:

#ifdef  MILKYWAY_BH
/* Hung Vu Oct 28, 1994
   We simply do not understand source routing
   just return ICMP_UNREACH and SRCFAIL
*/
                        type = ICMP_UNREACH;
                        code = ICMP_UNREACH_SRCFAIL;
                        goto bad;
#endif

                        if ((off = cp[IPOPT_OFFSET]) < IPOPT_MINOFF) {
                                code = &cp[IPOPT_OFFSET] - (u_char *)ip;
                                goto bad;
                        }
                        ipaddr.sin_addr = ip->ip_dst;
                        ia = (struct in_ifaddr *)
                                ifa_ifwithaddr((struct sockaddr *)&ipaddr);
                        if (ia == 0) {
                                if (opt == IPOPT_SSRR) {
                                        type = ICMP_UNREACH;
                                        code = ICMP_UNREACH_SRCFAIL;
                                        goto bad;
```

```
                        }
                        /*
                         * Loose routing, and not at next destination
                         * yet; nothing to do except forward.
                         */
                        break;
                }
                off--;                  /* 0 origin */
                if (off > optlen - sizeof(struct in_addr)) {
                        /*
                         * End of source route.  Should be for us.
                         */
                        save_rte(cp, ip->ip_src);
                        break;
                }
                /*
                 * locate outgoing interface
                 */
                bcopy((caddr_t)(cp + off), (caddr_t)&ipaddr.sin_addr,
                    sizeof(ipaddr.sin_addr));
                if (opt == IPOPT_SSRR) {
#define INA     struct in_ifaddr *
#define SA      struct sockaddr *
                    if ((ia = (INA)ifa_ifwithdstaddr((SA)&ipaddr)) == 0
                        ia = in_iaonnetof(in_netof(ipaddr.sin_addr));
                } else
                        ia = ip_rtaddr(ipaddr.sin_addr);
                if (ia == 0) {
                        type = ICMP_UNREACH;
                        code = ICMP_UNREACH_SRCFAIL;
                        goto bad;
                }
                ip->ip_dst = ipaddr.sin_addr;
                bcopy((caddr_t)&(IA_SIN(ia)->sin_addr),
                    (caddr_t)(cp + off), sizeof(struct in_addr));
                cp[IPOPT_OFFSET] += sizeof(struct in_addr);
                forward = 1;
                break;

        case IPOPT_RR:
                if ((off = cp[IPOPT_OFFSET]) < IPOPT_MINOFF) {
                        code = &cp[IPOPT_OFFSET] - (u_char *)ip;
                        goto bad;
                }
                /*
                 * If no space remains, ignore.
                 */
                off--;                  /* 0 origin */
                if (off > optlen - sizeof(struct in_addr))
                        break;
                bcopy((caddr_t)(&ip->ip_dst), (caddr_t)&ipaddr.sin_addr
                    sizeof(ipaddr.sin_addr));
                /*
                 * locate outgoing interface; if we're the destination,
                 * use the incoming interface (should be same).
                 */
                if ((ia = (INA)ifa_ifwithaddr((SA)&ipaddr)) == 0 &&
                    (ia = ip_rtaddr(ipaddr.sin_addr)) == 0) {
                        type = ICMP_UNREACH;
                        code = ICMP_UNREACH_HOST;
                        goto bad;
                }
                bcopy((caddr_t)&(IA_SIN(ia)->sin_addr),
                    (caddr_t)(cp + off), sizeof(struct in_addr));
                cp[IPOPT_OFFSET] += sizeof(struct in_addr);
                break;
```

```
                case IPOPT_TS:
                        code = cp - (u_char *)ip;
                        ipt = (struct ip_timestamp *)cp;
                        if (ipt->ipt_len < 5)
                                goto bad;
                        if (ipt->ipt_ptr > ipt->ipt_len - sizeof (long)) {
                                if (++ipt->ipt_oflw == 0)
                                        goto bad;
                                break;
                        }
                        sin = (struct in_addr *)(cp + ipt->ipt_ptr - 1);
                        switch (ipt->ipt_flg) {

                        case IPOPT_TS_TSONLY:
                                break;

                        case IPOPT_TS_TSANDADDR:
                                if (ipt->ipt_ptr + sizeof(n_time) +
                                    sizeof(struct in_addr) > ipt->ipt_len)
                                        goto bad;
                                ia = ifptoia(m->m_pkthdr.rcvif);
                                bcopy((caddr_t)&IA_SIN(ia)->sin_addr,
                                    (caddr_t)sin, sizeof(struct in_addr));
                                ipt->ipt_ptr += sizeof(struct in_addr);
                                break;

                        case IPOPT_TS_PRESPEC:
                                if (ipt->ipt_ptr + sizeof(n_time) +
                                    sizeof(struct in_addr) > ipt->ipt_len)
                                        goto bad;
                                bcopy((caddr_t)sin, (caddr_t)&ipaddr.sin_addr,
                                    sizeof(struct in_addr));
                                if (ifa_ifwithaddr((SA)&ipaddr) == 0)
                                        continue;
                                ipt->ipt_ptr += sizeof(struct in_addr);
                                break;

                        default:
                                goto bad;
                        }
                        ntime = iptime();
                        bcopy((caddr_t)&ntime, (caddr_t)cp + ipt->ipt_ptr - 1,
                            sizeof(n_time));
                        ipt->ipt_ptr += sizeof(n_time);
                }
        }
        if (forward) {
                if (ipforward_srcrt == 0) {
                        type = ICMP_UNREACH;
                        code = ICMP_UNREACH_SRCFAIL;
                        goto bad;
                }
                ip_forward(m, 1);
                return (1);
        } else
                return (0);
bad:
        icmp_error(m, type, code);
        return (1);
}

/*
 * Given address of next destination (final or next hop),
 * return internet address info of interface to be used to get there.
 */
```

```
struct in_ifaddr *
ip_rtaddr(dst)
         struct in_addr dst;
{
         egister struct sockaddr_in *sin;

        sin = (struct sockaddr_in *) &ipforward_rt.ro_dst;

        if (ipforward_rt.ro_rt == 0 || dst.s_addr != sin->sin_addr.s_addr) {
                if (ipforward_rt.ro_rt) {
                        RTFREE(ipforward_rt.ro_rt);
                        ipforward_rt.ro_rt = 0;
                }
                sin->sin_family = AF_INET;
                sin->sin_len = sizeof(*sin);
                sin->sin_addr = dst;

                rtalloc(&ipforward_rt);
        }
        if (ipforward_rt.ro_rt == 0)
                return ((struct in_ifaddr *)0);
        return ((struct in_ifaddr *) ipforward_rt.ro_rt->rt_ifa);
}

/*
 * Save incoming source route for use in replies,
 * to be picked up later by ip_srcroute if the receiver is interested.
 */
save_rte(option, dst)
        u_char *option;
        struct in_addr dst;
{
        unsigned olen;

        olen = option[IPOPT_OLEN];
#ifdef DEBUG
        if (ipprintfs)
                printf("save_rte: olen %d\n", olen);
#endif
        if (olen > sizeof(ip_srcrt) - (1 + sizeof(dst)))
                return;
        bcopy((caddr_t)option, (caddr_t)ip_srcrt.srcopt, olen);
        ip_nhops = (olen - IPOPT_OFFSET - 1) / sizeof(struct in_addr);
        ip_srcrt.dst = dst;
}

/*
 * Retrieve incoming source route for use in replies,
 * in the same form used by setsockopt.
 * The first hop is placed before the options, will be removed later.
 */
struct mbuf *
ip_srcroute()
{
        register struct in_addr *p, *q;
        register struct mbuf *m;

        if (ip_nhops == 0)
                return ((struct mbuf *)0);
        m = m_get(M_DONTWAIT, MT_SOOPTS);
        if (m == 0)
                return ((struct mbuf *)0);

#define OPTSIZ  (sizeof(ip_srcrt.nop) + sizeof(ip_srcrt.srcopt))

        /* length is (nhops+1)*sizeof(addr) + sizeof(nop + srcrt header) */
```

```
        m->m_len = ip_nhops * sizeof(struct in_addr) + sizeof(struct in_addr) +
            OPTSIZ;
#ifdef DEBUG
        if (ipprintfs)
                printf("ip_srcroute: nhops %d mlen %d", ip_nhops, m->m_len);
#endif

        /*
         * First save first hop for return route
         */
        p = &ip_srcrt.route[ip_nhops - 1];
        *(mtod(m, struct in_addr *)) = *p--;
#ifdef DEBUG
        if (ipprintfs)
                printf(" hops %lx", ntohl(mtod(m, struct in_addr *)->s_addr));
#endif

        /*
         * Copy option fields and padding (nop) to mbuf.
         */
        ip_srcrt.nop = IPOPT_NOP;
        ip_srcrt.srcopt[IPOPT_OFFSET] = IPOPT_MINOFF;
        bcopy((caddr_t)&ip_srcrt.nop,
            mtod(m, caddr_t) + sizeof(struct in_addr), OPTSIZ);
        q = (struct in_addr *)(mtod(m, caddr_t) +
            sizeof(struct in_addr) + OPTSIZ);
#undef OPTSIZ
        /*
         * Record return path as an IP source route,
         * reversing the path (pointers are now aligned).
         */
        while (p >= ip_srcrt.route) {
#ifdef DEBUG
                if (ipprintfs)
                        printf(" %lx", ntohl(q->s_addr));
#endif
                *q++ = *p--;
        }
        /*
         * Last hop goes to final destination.
         */
        *q = ip_srcrt.dst;
#ifdef DEBUG
        if (ipprintfs)
                printf(" %lx\n", ntohl(q->s_addr));
#endif
        return (m);
}

/*
 * Strip out IP options, at higher
 * level protocol in the kernel.
 * Second argument is buffer to which options
 * will be moved, and return value is their length.
 * XXX should be deleted; last arg currently ignored.
 */
ip_stripoptions(m, mopt)
        register struct mbuf *m;
        struct mbuf *mopt;
{
        register int i;
        struct ip *ip = mtod(m, struct ip *);
        register caddr_t opts;
        int olen;

        olen = (ip->ip_hl<<2) - sizeof (struct ip);
```

```
		opts = (caddr_t)(ip + 1);
		i = m->m_len - (sizeof (struct ip) + olen);
		bcopy(opts  + olen, opts, (unsigned)i);
		m->m_len -= olen;
		if (m->m_flags & M_PKTHDR)
			m->m_pkthdr.len -= olen;
		ip->ip_hl = sizeof(struct ip) >> 2;
}

u_char inetctlerrmap[PRC_NCMDS] = {
	0,		0,		0,		0,
	0,		EMSGSIZE,	EHOSTDOWN,	EHOSTUNREACH,
	EHOSTUNREACH,	EHOSTUNREACH,	ECONNREFUSED,	ECONNREFUSED,
	EMSGSIZE,	EHOSTUNREACH,	0,		0,
	0,		0,		0,		0,
	ENOPROTOOPT
};

/*
 * Forward a packet.  If some error occurs return the sender
 * an icmp packet.  Note we can't always generate a meaningful
 * icmp message because icmp doesn't have a large enough repertoire
 * of codes and types.
 *
 * If not forwarding, just drop the packet.  This could be confusing
 * if ipforwarding was zero but some routing protocol was advancing
 * us as a gateway to somewhere.  However, we must let the routing
 * protocol deal with that.
 *
 * The srcrt parameter indicates whether the packet is being forwarded
 * via a source route.
 */
void
ip_forward(m, srcrt)
	struct mbuf *m;
	int srcrt;
{
	register struct ip *ip = mtod(m, struct ip *);
	register struct sockaddr_in *sin;
	register struct rtentry *rt;
	int error, type = 0, code;
	struct mbuf *mcopy;
	struct in_addr dest;

	dest.s_addr = 0;
#ifdef DEBUG
	if (ipprintfs)
		printf("forward: src %x dst %x ttl %x\n", ip->ip_src,
			ip->ip_dst, ip->ip_ttl);
#endif
	if (m->m_flags & M_BCAST || in_canforward(ip->ip_dst) == 0) {
		ipstat.ips_cantforward++;
		m_freem(m);
		return;
	}
	HTONS(ip->ip_id);
	if (ip->ip_ttl <= IPTTLDEC) {
		icmp_error(m, ICMP_TIMXCEED, ICMP_TIMXCEED_INTRANS, dest);
		return;
	}
	ip->ip_ttl -= IPTTLDEC;

	sin = (struct sockaddr_in *)&ipforward_rt.ro_dst;
	if ((rt = ipforward_rt.ro_rt) == 0 ||
	    ip->ip_dst.s_addr != sin->sin_addr.s_addr) {
		if (ipforward_rt.ro_rt) {
```

```
                        RTFREE(ipforward_rt.ro_rt);
                        ipforward_rt.ro_rt = 0;
                }
                sin->sin_family = AF_INET;
                sin->sin_len = sizeof(*sin);
                sin->sin_addr = ip->ip_dst;

                rtalloc(&ipforward_rt);
                if (ipforward_rt.ro_rt == 0) {
                        icmp_error(m, ICMP_UNREACH, ICMP_UNREACH_HOST, dest);
                        return;
                }
                rt = ipforward_rt.ro_rt;
        }

        /*
         * Save at most 64 bytes of the packet in case
         * we need to generate an ICMP message to the src.
         */
        mcopy = m_copy(m, 0, imin((int)ip->ip_len, 64));

#ifdef GATEWAY
        ip_ifmatrix[rt->rt_ifp->if_index +
             if_index * m->m_pkthdr.rcvif->if_index]++;
#endif
        /*
         * If forwarding packet using same interface that it came in on,
         * perhaps should send a redirect to sender to shortcut a hop.
         * Only send redirect if source is sending directly to us,
         * and if packet was not source routed (or has any options).
         * Also, don't send redirect if forwarding using a default route
         * or a route modified by a redirect.
         */
#define satosin(sa)     ((struct sockaddr_in *)(sa))
        if (rt->rt_ifp == m->m_pkthdr.rcvif &&
            (rt->rt_flags & (RTF_DYNAMIC|RTF_MODIFIED)) == 0 &&
            satosin(rt_key(rt))->sin_addr.s_addr != 0 &&
            ipsendredirects && !srcrt) {
                struct in_ifaddr *ia;
                u_long src = ntohl(ip->ip_src.s_addr);
                u_long dst = ntohl(ip->ip_dst.s_addr);

                if ((ia = ifptoia(m->m_pkthdr.rcvif)) &&
                   (src & ia->ia_subnetmask) == ia->ia_subnet) {
                    if (rt->rt_flags & RTF_GATEWAY)
                        dest = satosin(rt->rt_gateway)->sin_addr;
                    else
                        dest = ip->ip_dst;
                    /*
                     * If the destination is reached by a route to host,
                     * is on a subnet of a local net, or is directly
                     * on the attached net (!), use host redirect.
                     * (We may be the correct first hop for other subnets.)
                     */
#define RTA(rt) ((struct in_ifaddr *)(rt->rt_ifa))
                    type = ICMP_REDIRECT;
                    if ((rt->rt_flags & RTF_HOST) ||
                        (rt->rt_flags & RTF_GATEWAY) == 0)
                            code = ICMP_REDIRECT_HOST;
                    else if (RTA(rt)->ia_subnetmask != RTA(rt)->ia_netmask &&
                        (dst & RTA(rt)->ia_netmask) ==  RTA(rt)->ia_net)
                            code = ICMP_REDIRECT_HOST;
                    else
                            code = ICMP_REDIRECT_NET;
#ifdef DEBUG
                    if (ipprintfs)
```

```
                        printf("redirect (%d) to %x\n", code, dest.s_addr);
#endif
                }
        }

#ifdef DIRECTED_BROADCAST
        error = ip_output(m, (struct mbuf *)0, &ipforward_rt,
            IP_FORWARDING | IP_ALLOWBROADCAST);
#else
        error = ip_output(m, (struct mbuf *)0, &ipforward_rt, IP_FORWARDING);
#endif
        if (error)
                ipstat.ips_cantforward++;
        else {
                ipstat.ips_forward++;
                if (type)
                        ipstat.ips_redirectsent++;
                else {
                        if (mcopy)
                                m_freem(mcopy);
                        return;
                }
        }
        if (mcopy == NULL)
                return;
        switch (error) {

        case 0:                         /* forwarded, but need redirect */
                /* type, code set above */
                break;

        case ENETUNREACH:               /* shouldn't happen, checked above */
        case EHOSTUNREACH:
        case ENETDOWN:
        case EHOSTDOWN:
        default:
                type = ICMP_UNREACH;
                code = ICMP_UNREACH_HOST;
                break;

        case EMSGSIZE:
                type = ICMP_UNREACH;
                code = ICMP_UNREACH_NEEDFRAG;
                ipstat.ips_cantfrag++;
                break;

        case ENOBUFS:
                type = ICMP_SOURCEQUENCH;
                code = 0;
                break;
        }
        icmp_error(mcopy, type, code, dest);
}
```

```
/*      BSDI $Id: in_pcb.c,v 1.1.1.1.4.1 1994/11/16 15:11:14 mcr Exp $  */

/*
 * Copyright (c) 1982, 1986, 1991 Regents of the University of California.
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 * 3. All advertising materials mentioning features or use of this software
 *    must display the following acknowledgement:
 *      This product includes software developed by the University of
 *      California, Berkeley and its contributors.
 * 4. Neither the name of the University nor the names of its contributors
 *    may be used to endorse or promote products derived from this software
 *    without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS ``AS IS'' AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 *
 *      @(#)in_pcb.c    7.14 (Berkeley) 4/20/91
 */

#include "param.h"
#include "systm.h"
#include "malloc.h"
#include "mbuf.h"
#include "proc.h"
#include "protosw.h"
#include "socket.h"
#include "socketvar.h"
#include "ioctl.h"

#include "../net/if.h"
#include "../net/route.h"

#include "in.h"
#include "in_systm.h"
#include "ip.h"
#include "in_pcb.h"
#include "in_var.h"
#include "ip_var.h"

#define ANY_PORT        htons(59813)
#define OPENWIN_PORT    htons(2000)
#define XWINDOW_PORT    htons(6000)
#define SYSLOGD_PORT    htons(514)

struct  in_addr zeroin_addr;

struct inpcb *
```

```
in_pcblokup(head, faddr, fport, laddr, lport, flags)
        struct inpcb *head;
        struct in_addr faddr, laddr;
        u_short fport, lport;
        int flags;
{
        register struct inpcb *inp, *match = 0;
        int matchwild = 3, wildcard;

        for (inp = head->inp_next; inp != head; inp = inp->inp_next) {
                if (inp->inp_lport != lport)
                        continue;
                wildcard = 0;
                if (inp->inp_laddr.s_addr != INADDR_ANY) {
                        if (laddr.s_addr == INADDR_ANY)
                                wildcard++;
                        else if (inp->inp_laddr.s_addr != laddr.s_addr)
                                continue;
                } else {
                        if (laddr.s_addr != INADDR_ANY)
                                wildcard++;
                }
                if (inp->inp_faddr.s_addr != INADDR_ANY) {
                        if (faddr.s_addr == INADDR_ANY)
                                wildcard++;
                        else if (inp->inp_faddr.s_addr != faddr.s_addr ||
                            inp->inp_fport != fport)
                                continue;
                } else {
                        if (faddr.s_addr != INADDR_ANY)
                                wildcard++;
                }
                if (wildcard && (flags & INPLOOKUP_WILDCARD) == 0)
                        continue;
                if (wildcard < matchwild) {
                        match = inp;
                        matchwild = wildcard;
                        if (matchwild == 0)
                                break;
                }
        }
        return (match);
}

in_pcballoc(so, head)
        struct socket *so;
        struct inpcb *head;
{
        struct mbuf *m;
        register struct inpcb *inp;

        m = m_getclr(M_DONTWAIT, MT_PCB);
        if (m == NULL)
                return (ENOBUFS);
        inp = mtod(m, struct inpcb *);
        inp->inp_head = head;
        inp->inp_socket = so;
        insque(inp, head);
        so->so_pcb = (caddr_t)inp;
        return (0);
}

in_pcbbind(inp, nam)
        register struct inpcb *inp;
        struct mbuf *nam;
{
```

```
        register struct socket *so = inp->inp_socket;
        register struct inpcb *head = inp->inp_head;
        register struct sockaddr_in *sin;
        struct proc *p = curproc;               /* XXX */
        u_short lport = 0;
        int wild = 0;

        if (in_ifaddr == 0)
                return (EADDRNOTAVAIL);
        if (inp->inp_lport || inp->inp_laddr.s_addr != INADDR_ANY)
                return (EINVAL);

        if ((so->so_options & SO_REUSEADDR) == 0 &&
            ((so->so_proto->pr_flags & PR_CONNREQUIRED) == 0 ||
             (so->so_options & SO_ACCEPTCONN) == 0))
                wild = INPLOOKUP_WILDCARD;

        if (nam == 0)
                goto noname;
        sin = mtod(nam, struct sockaddr_in *);
        if (nam->m_len != sizeof (*sin))
                return (EINVAL);
        if (sin->sin_addr.s_addr != INADDR_ANY) {
                int tport = sin->sin_port;

                sin->sin_port = 0;              /* yech... */
                if (ifa_ifwithaddr((struct sockaddr *)sin) == 0)
                        return (EADDRNOTAVAIL);
                sin->sin_port = tport;
        }
        lport = sin->sin_port;
        if (lport) {
                u_short aport = ntohs(lport);

                /* GROSS */
                if (aport < IPPORT_RESERVED && suser(p->p_ucred, &p->p_acflag))
                        return (EACCES);
                if (in_pcblokup(head,
                    zeroin_addr, 0, sin->sin_addr, lport, wild))
                        return (EADDRINUSE);
        }
        inp->inp_laddr = sin->sin_addr;
noname:
        if (lport == 0)
                do {
                        if (head->inp_lport++ < IPPORT_RESERVED ||
                            head->inp_lport > IPPORT_USERRESERVED)
                                head->inp_lport = IPPORT_RESERVED;
                        lport = htons(head->inp_lport);
#ifdef  MILKYWAY_BH
                        if (lport == ANY_PORT) lport++;
#endif
                } while (in_pcblokup(head,
                            zeroin_addr, 0, inp->inp_laddr, lport, wild));
        inp->inp_lport = lport;
        return (0);
}

/*
 * Connect from a socket to a specified address.
 * Both address and port must be specified in argument sin.
 * If don't have a local address for this socket yet,
 * then pick one.
 */
in_pcbconnect(inp, nam)
        register struct inpcb *inp;
```

```
        struct mbuf *nam;
{
        struct in_ifaddr *ia;
        struct sockaddr_in *ifaddr;
        register struct sockaddr_in *sin = mtod(nam, struct sockaddr_in *);

        if (nam->m_len != sizeof (*sin))
                return (EINVAL);
        if (sin->sin_family != AF_INET)
                return (EAFNOSUPPORT);
        if (sin->sin_port == 0)
                return (EADDRNOTAVAIL);
        if (in_ifaddr) {
                /*
                 * If the destination address is INADDR_ANY,
                 * use the primary local address.
                 * If the supplied address is INADDR_BROADCAST,
                 * and the primary interface supports broadcast,
                 * choose the broadcast address for that interface.
                 */
#define satosin(sa)     ((struct sockaddr_in *)(sa))
                if (sin->sin_addr.s_addr == INADDR_ANY)
                    sin->sin_addr = IA_SIN(in_ifaddr)->sin_addr;
                else if (sin->sin_addr.s_addr == (u_long)INADDR_BROADCAST &&
                  (in_ifaddr->ia_ifp->if_flags & IFF_BROADCAST))
                    sin->sin_addr = satosin(&in_ifaddr->ia_broadaddr)->sin_addr;
        }
        if (inp->inp_laddr.s_addr == INADDR_ANY) {
                register struct route *ro;
                struct ifnet *ifp;

                ia = (struct in_ifaddr *)0;
                /*
                 * If route is known or can be allocated now,
                 * our src addr is taken from the i/f, else punt.
                 */
                ro = &inp->inp_route;
                if (ro->ro_rt &&
                    (satosin(&ro->ro_dst)->sin_addr.s_addr !=
                        sin->sin_addr.s_addr ||
                    inp->inp_socket->so_options & SO_DONTROUTE)) {
                        RTFREE(ro->ro_rt);
                        ro->ro_rt = (struct rtentry *)0;
                }
                if ((inp->inp_socket->so_options & SO_DONTROUTE) == 0 && /*XXX*/
                    (ro->ro_rt == (struct rtentry *)0 ||
                    ro->ro_rt->rt_ifp == (struct ifnet *)0)) {
                        /* No route yet, so try to acquire one */
                        ro->ro_dst.sa_family = AF_INET;
                        ro->ro_dst.sa_len = sizeof(struct sockaddr_in);
                        ((struct sockaddr_in *) &ro->ro_dst)->sin_addr =
                                sin->sin_addr;
                        rtalloc(ro);
                }
                /*
                 * If we found a route, use the address
                 * corresponding to the outgoing interface
                 * unless it is the loopback (in case a route
                 * to our address on another net goes to loopback).
                 */
                if (ro->ro_rt && (ifp = ro->ro_rt->rt_ifp) &&
                    (ifp->if_flags & IFF_LOOPBACK) == 0 &&
                    (ia = (struct in_ifaddr *)ro->ro_rt->rt_ifa) == 0)
                        for (ia = in_ifaddr; ia; ia = ia->ia_next)
                                if (ia->ia_ifp == ifp)
                                        break;
```

```
                if (ia == 0) {
                        int fport = sin->sin_port;

                        sin->sin_port = 0;
                        ia = (struct in_ifaddr *)
                            ifa_ifwithdstaddr((struct sockaddr *)sin);
                        sin->sin_port = fport;
                        if (ia == 0)
                                ia = in_iaonnetof(in_netof(sin->sin_addr));
                        if (ia == 0)
                                ia = in_ifaddr;
                        if (ia == 0)
                                return (EADDRNOTAVAIL);
                }
#ifdef MULTICAST
                /*
                 * If the destination address is multicast and an outgoing
                 * interface has been set as a multicast option, use the
                 * address of that interface as our source address.
                 */
                if (IN_MULTICAST(ntohl(sin->sin_addr.s_addr)) &&
                    inp->inp_moptions != NULL) {
                        struct ip_moptions *imo;
                        struct ifnet *ifp;

                        imo = inp->inp_moptions;
                        if (imo->imo_multicast_ifp != NULL) {
                                ifp = imo->imo_multicast_ifp;
                                for (ia = in_ifaddr; ia; ia = ia->ia_next)
                                        if (ia->ia_ifp == ifp)
                                                break;
                                if (ia == 0)
                                        return (EADDRNOTAVAIL);
                        }
                }
#endif
                ifaddr = (struct sockaddr_in *)&ia->ia_addr;
        }
        if (in_pcblokup(inp->inp_head,
            sin->sin_addr,
            sin->sin_port,
            inp->inp_laddr.s_addr ? inp->inp_laddr : ifaddr->sin_addr,
            inp->inp_lport,
            0))
                return (EADDRINUSE);
        if (inp->inp_laddr.s_addr == INADDR_ANY) {
                if (inp->inp_lport == 0)
                        (void)in_pcbbind(inp, (struct mbuf *)0);
                inp->inp_laddr = ifaddr->sin_addr;
        }
        inp->inp_faddr = sin->sin_addr;
        inp->inp_fport = sin->sin_port;
        return (0);
}

in_pcbdisconnect(inp)
        struct inpcb *inp;
{

        inp->inp_faddr.s_addr = INADDR_ANY;
        inp->inp_fport = 0;
        if (inp->inp_socket->so_state & SS_NOFDREF)
                in_pcbdetach(inp);
}

in_pcbdetach(inp)
```

```
        struct inpcb *inp;
{
        struct socket *so = inp->inp_socket;

        so->so_pcb = 0;
        sofree(so);
        if (inp->inp_options)
                (void)m_free(inp->inp_options);
        if (inp->inp_route.ro_rt)
                rtfree(inp->inp_route.ro_rt);
#ifdef MULTICAST
        if (inp->inp_moptions)
                ip_freemoptions(inp->inp_moptions);
#endif
        remque(inp);
        (void) m_free(dtom(inp));
}

in_setsockaddr(inp, nam)
        register struct inpcb *inp;
        struct mbuf *nam;
{
        register struct sockaddr_in *sin;

        nam->m_len = sizeof (*sin);
        sin = mtod(nam, struct sockaddr_in *);
        bzero((caddr_t)sin, sizeof (*sin));
        sin->sin_family = AF_INET;
        sin->sin_len = sizeof(*sin);
        sin->sin_port = inp->inp_lport;
        sin->sin_addr = inp->inp_laddr;
}

in_setpeeraddr(inp, nam)
        struct inpcb *inp;
        struct mbuf *nam;
{
        register struct sockaddr_in *sin;

        nam->m_len = sizeof (*sin);
        sin = mtod(nam, struct sockaddr_in *);
        bzero((caddr_t)sin, sizeof (*sin));
        sin->sin_family = AF_INET;
        sin->sin_len = sizeof(*sin);
        sin->sin_port = inp->inp_fport;
        sin->sin_addr = inp->inp_faddr;
}

/*
 * Pass some notification to all connections of a protocol
 * associated with address dst.  The local address and/or port numbers
 * may be specified to limit the search.  The "usual action" will be
 * taken, depending on the ctlinput cmd.  The caller must filter any
 * cmds that are uninteresting (e.g., no error in the map).
 * Call the protocol specific routine (if any) to report
 * any errors for each matching socket.
 *
 * Must be called at splnet.
 */
in_pcbnotify(head, dst, fport, laddr, lport, cmd, notify)
        struct inpcb *head;
        struct sockaddr *dst;
        u_short fport, lport;
        struct in_addr laddr;
        int cmd, (*notify)();
{
```

```
        register struct inpcb *inp, *oinp;
        struct in_addr faddr;
        int errno;
        int in_rtchange();
        extern u_char inetctlerrmap[];

        if ((unsigned)cmd > PRC_NCMDS || dst->sa_family != AF_INET)
                return;
        faddr = ((struct sockaddr_in *)dst)->sin_addr;
        if (faddr.s_addr == INADDR_ANY)
                return;

        /*
         * Redirects go to all references to the destination,
         * and use in_rtchange to invalidate the route cache.
         * Dead host indications: notify all references to the destination.
         * Otherwise, if we have knowledge of the local port and address,
         * deliver only to that socket.
         */
        if (PRC_IS_REDIRECT(cmd) || cmd == PRC_HOSTDEAD) {
                fport = 0;
                lport = 0;
                laddr.s_addr = 0;
                if (cmd != PRC_HOSTDEAD)
                        notify = in_rtchange;
        }
        errno = inetctlerrmap[cmd];
        for (inp = head->inp_next; inp != head;) {
                if (inp->inp_faddr.s_addr != faddr.s_addr ||
                    inp->inp_socket == 0 ||
                    (lport && inp->inp_lport != lport) ||
                    (laddr.s_addr && inp->inp_laddr.s_addr != laddr.s_addr) ||
                    (fport && inp->inp_fport != fport)) {
                        inp = inp->inp_next;
                        continue;
                }
                oinp = inp;
                inp = inp->inp_next;
                if (notify)
                        (*notify)(oinp, errno);
        }
}

/*
 * Check for alternatives when higher level complains
 * about service problems.  For now, invalidate cached
 * routing information.  If the route was created dynamically
 * (by a redirect), time to try a default gateway again.
 */
in_losing(inp)
        struct inpcb *inp;
{
        register struct rtentry *rt;

        if ((rt = inp->inp_route.ro_rt)) {
                rt_missmsg(RTM_LOSING, &inp->inp_route.ro_dst,
                            rt->rt_gateway, (struct sockaddr *)rt_mask(rt),
                            (struct sockaddr *)0, rt->rt_flags, 0);
                if (rt->rt_flags & RTF_DYNAMIC)
                        (void) rtrequest(RTM_DELETE, rt_key(rt),
                                rt->rt_gateway, rt_mask(rt), rt->rt_flags,
                                (struct rtentry **)0);
                inp->inp_route.ro_rt = 0;
                rtfree(rt);
                /*
                 * A new route can be allocated
```

```
                 * the next time output is attempted.
                 */
        }
}

/*
 * After a routing change, flush old routing
 * and allocate a (hopefully) better one.
 */
in_rtchange(inp)
        register struct inpcb *inp;
{
        if (inp->inp_route.ro_rt) {
                rtfree(inp->inp_route.ro_rt);
                inp->inp_route.ro_rt = 0;
                /*
                 * A new route can be allocated the next time
                 * output is attempted.
                 */
        }
}

struct inpcb *
in_pcblookup(head, faddr, fport, laddr, lport, flags)
        struct inpcb *head;
        struct in_addr faddr, laddr;
        u_short fport, lport;
        int flags;
{
        register struct inpcb *inp, *match = 0;
        int matchwild = 3, wildcard;

        for (inp = head->inp_next; inp != head; inp = inp->inp_next) {
                if (inp->inp_lport != lport)
#ifdef  MILKYWAY_BH
                        if ((inp->inp_lport != ANY_PORT) || (laddr.s_addr == INA
DDR_LOOPBACK))
#endif
                                continue;
#ifdef  MILKYWAY_BH
                if ( ((lport == OPENWIN_PORT) || (lport == XWINDOW_PORT) || (lpo
rt == SYSLOGD_PORT)) && (laddr.s_addr != INADDR_LOOPBACK) ) continue;
#endif
                wildcard = 0;
                if (inp->inp_laddr.s_addr != INADDR_ANY) {
                        if (laddr.s_addr == INADDR_ANY)
                                wildcard++;
                        else if (inp->inp_laddr.s_addr != laddr.s_addr)
                                continue;
                } else {
                        if (laddr.s_addr != INADDR_ANY)
                                wildcard++;
                }
                if (inp->inp_faddr.s_addr != INADDR_ANY) {
                        if (faddr.s_addr == INADDR_ANY)
                                wildcard++;
                        else if (inp->inp_faddr.s_addr != faddr.s_addr ||
                            inp->inp_fport != fport)
                                continue;
                } else {
                        if (faddr.s_addr != INADDR_ANY)
                                wildcard++;
                }
                if (wildcard && (flags & INPLOOKUP_WILDCARD) == 0)
                        continue;
```

```
                if (wildcard < matchwild) {
                        match = inp;
                        matchwild = wildcard;
                        if (matchwild == 0)
                                break;
                }
        }
        return (match);
}
```

```
/*      BSDI $Id: ip_icmp.c,v 1.1.1.1.4.1 1994/11/16 15:11:15 mcr Exp $ */

/*
 * Copyright (c) 1982, 1986, 1989, 1993
 *       egents of the University of California.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 * 3. All advertising materials mentioning features or use of this software
 *    must display the following acknowledgement:
 *      This product includes software developed by the University of
 *      California, Berkeley and its contributors.
 * 4. Neither the name of the University nor the names of its contributors
 *    may be used to endorse or promote products derived from this software
 *    without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS ``AS IS'' AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 *
 *      @(#)ip_icmp.c   7.15 (Berkeley) 4/20/91
 */

#include "param.h"
#include "systm.h"
#include "malloc.h"
#include "mbuf.h"
#include "protosw.h"
#include "socket.h"
#include "time.h"
#include "kernel.h"

#include "../net/route.h"
#include "../net/if.h"

#include "in.h"
#include "in_systm.h"
#include "in_var.h"
#include "ip.h"
#include "ip_icmp.h"
#include "icmp_var.h"

/*
 * ICMP routines: error generation, receive packet processing, and
 * routines to turnaround packets back to the originator, and
 * host table maintenance routines.
 */
#ifdef ICMPPRINTFS
int     icmpprintfs = 0;
#endif

extern  struct protosw inetsw[];
```

```
/*
 * Generate an error packet of type error
 * in response to bad packet ip.
 */
/*VARARGS3*/
icmp_error(n, type, code, dest)
        struct mbuf *n;
        int type, code;
        struct in_addr dest;
{
        register struct ip *oip = mtod(n, struct ip *), *nip;
        register unsigned oiplen = oip->ip_hl << 2;
        register struct icmp *icp;
        register struct mbuf *m;
        unsigned icmplen;

#ifdef ICMPPRINTFS
        if (icmpprintfs)
                printf("icmp_error(%x, %d, %d)\n", oip, type, code);
#endif
        /*      Hung Vu Oct 28, 1994
                Just ignore ICMP REDIRECT
        */
#ifdef  MILKYWAY_BH
        if (type == ICMP_REDIRECT) goto freeit;
        else icmpstat.icps_error++;
#else
        if (type != ICMP_REDIRECT)
                icmpstat.icps_error++;
#endif

        /*
         * Don't send error if not the first fragment of message.
         * Don't error if the old packet protocol was ICMP
         * error message, only known informational types.
         */
        if (oip->ip_off &~ (IP_MF|IP_DF))
                goto freeit;
        if (oip->ip_p == IPPROTO_ICMP && type != ICMP_REDIRECT &&
          n->m_len >= oiplen + ICMP_MINLEN &&
          !ICMP_INFOTYPE(((struct icmp *)((caddr_t)oip + oiplen))->icmp_type))
                icmpstat.icps_oldicmp++;
                goto freeit;
        }
        /* Don't send error in response to a multicast or broadcast packet */
        if (n->m_flags & (M_MCAST | M_BCAST))
                goto freeit;
        /*
         * First, formulate icmp message
         */
        m = m_gethdr(M_DONTWAIT, MT_HEADER);
        if (m == NULL)
                goto freeit;
        icmplen = oiplen + min(8, oip->ip_len);
        m->m_len = icmplen + ICMP_MINLEN;
        MH_ALIGN(m, m->m_len);
        icp = mtod(m, struct icmp *);
        if ((u_int)type > ICMP_MAXTYPE)
                panic("icmp_error");
        icmpstat.icps_outhist[type]++;
        icp->icmp_type = type;
        if (type == ICMP_REDIRECT)
                icp->icmp_gwaddr = dest;
        else
                icp->icmp_void = 0;
```

```
        if (type == ICMP_PARAMPROB) {
                icp->icmp_pptr = code;
                code = 0;
        }
         cp->icmp_code = code;
        bcopy((caddr_t)oip, (caddr_t)&icp->icmp_ip, icmplen);
        nip = &icp->icmp_ip;
        nip->ip_len = htons((u_short)(nip->ip_len + oiplen));

        /*
         * Now, copy old ip header (without options)
         * in front of icmp message.
         */
        if (m->m_data - sizeof(struct ip) < m->m_pktdat)
                panic("icmp len");
        m->m_data -= sizeof(struct ip);
        m->m_len += sizeof(struct ip);
        m->m_pkthdr.len = m->m_len;
        m->m_pkthdr.rcvif = n->m_pkthdr.rcvif;
        nip = mtod(m, struct ip *);
        bcopy((caddr_t)oip, (caddr_t)nip, oiplen);
        nip->ip_len = m->m_len;
        nip->ip_hl = sizeof(struct ip) >> 2;
        nip->ip_p = IPPROTO_ICMP;
        icmp_reflect(m);

freeit:
        m_freem(n);
}

static struct sockproto icmproto = { AF_INET, IPPROTO_ICMP };
static struct sockaddr_in icmpsrc = { sizeof (struct sockaddr_in), AF_INET };
static struct sockaddr_in icmpdst = { sizeof (struct sockaddr_in), AF_INET };
static struct sockaddr_in icmpgw = { sizeof (struct sockaddr_in), AF_INET };
struct sockaddr_in icmpmask = { 8, 0 };
struct in_ifaddr *ifptoia();

/*
 * Process a received ICMP message.
 */
icmp_input(m, hlen)
        register struct mbuf *m;
        int hlen;
{
        register struct icmp *icp;
        register struct ip *ip = mtod(m, struct ip *);
        int icmplen = ip->ip_len;
        register int i;
        struct in_ifaddr *ia;
        int (*ctlfunc)(), code;
        extern u_char ip_protox[];
        extern struct in_addr in_makeaddr();

        /*
         * Locate icmp structure in mbuf, and check
         * that not corrupted and of at least minimum length.
         */
#ifdef ICMPPRINTFS
        if (icmpprintfs)
                printf("icmp_input from %x, len %d\n", ip->ip_src, icmplen);
#endif
        if (icmplen < ICMP_MINLEN) {
                icmpstat.icps_tooshort++;
                goto freeit;
        }
        i = hlen + MIN(icmplen, ICMP_ADVLENMIN);
```

```
        if (m->m_len < i && (m = m_pullup(m, i)) == 0)  {
                icmpstat.icps_tooshort++;
                return;
        }
        p = mtod(m, struct ip *);
        m->m_len -= hlen;
        m->m_data += hlen;
        icp = mtod(m, struct icmp *);
        if (in_cksum(m, icmplen)) {
                icmpstat.icps_checksum++;
                goto freeit;
        }
        m->m_len += hlen;
        m->m_data -= hlen;

#ifdef ICMPPRINTFS
        /*
         * Message type specific processing.
         */
        if (icmpprintfs)
                printf("icmp_input, type %d code %d\n", icp->icmp_type,
                    icp->icmp_code);
#endif
        if (icp->icmp_type > ICMP_MAXTYPE)
                goto raw;
        icmpstat.icps_inhist[icp->icmp_type]++;
        code = icp->icmp_code;
        switch (icp->icmp_type) {

        case ICMP_UNREACH:
                if (code > 5)
                        goto badcode;
                code += PRC_UNREACH_NET;
                goto deliver;

        case ICMP_TIMXCEED:
                if (code > 1)
                        goto badcode;
                code += PRC_TIMXCEED_INTRANS;
                goto deliver;

        case ICMP_PARAMPROB:
                if (code)
                        goto badcode;
                code = PRC_PARAMPROB;
                goto deliver;

        case ICMP_SOURCEQUENCH:
                if (code)
                        goto badcode;
                code = PRC_QUENCH;
        deliver:
                /*
                 * Problem with datagram; advise higher level routines.
                 */
                if (icmplen < ICMP_ADVLENMIN || icmplen < ICMP_ADVLEN(icp) ||
                    icp->icmp_ip.ip_hl < (sizeof(struct ip) >> 2)) {
                        icmpstat.icps_badlen++;
                        goto freeit;
                }
                NTOHS(icp->icmp_ip.ip_len);
#ifdef ICMPPRINTFS
                if (icmpprintfs)
                        printf("deliver to protocol %d\n", icp->icmp_ip.ip_p);
#endif
                icmpsrc.sin_addr = icp->icmp_ip.ip_dst;
```

```
                if (ctlfunc = inetsw[ip_protox[icp->icmp_ip.ip_p]].pr_ctlinput)
                        (*ctlfunc)(code, (struct sockaddr *)&icmpsrc,
                            (caddr_t) &icp->icmp_ip);
                break;

        badcode:
                icmpstat.icps_badcode++;
                break;

        case ICMP_ECHO:
                icp->icmp_type = ICMP_ECHOREPLY;
                goto reflect;

        case ICMP_TSTAMP:
                if (icmplen < ICMP_TSLEN) {
                        icmpstat.icps_badlen++;
                        break;
                }
                icp->icmp_type = ICMP_TSTAMPREPLY;
                icp->icmp_rtime = iptime();
                icp->icmp_ttime = icp->icmp_rtime;      /* bogus, do later! */
                goto reflect;

        case ICMP_IREQ:
#define satosin(sa)     ((struct sockaddr_in *)(sa))
                if (in_netof(ip->ip_src) == 0 &&
                    (ia = ifptoia(m->m_pkthdr.rcvif)))
                        ip->ip_src = in_makeaddr(in_netof(IA_SIN(ia)->sin_addr),
                            in_lnaof(ip->ip_src));
                icp->icmp_type = ICMP_IREQREPLY;
                goto reflect;

        case ICMP_MASKREQ:
                if (icmplen < ICMP_MASKLEN ||
                    (ia = ifptoia(m->m_pkthdr.rcvif)) == 0)
                        break;
                icp->icmp_type = ICMP_MASKREPLY;
                icp->icmp_mask = ia->ia_sockmask.sin_addr.s_addr;
                if (ip->ip_src.s_addr == 0) {
                        if (ia->ia_ifp->if_flags & IFF_BROADCAST)
                            ip->ip_src = satosin(&ia->ia_broadaddr)->sin_addr;
                        else if (ia->ia_ifp->if_flags & IFF_POINTOPOINT)
                            ip->ip_src = satosin(&ia->ia_dstaddr)->sin_addr;
                }
        reflect:
                ip->ip_len += hlen;     /* since ip_input deducts this */
                icmpstat.icps_reflect++;
                icmpstat.icps_outhist[icp->icmp_type]++;
                icmp_reflect(m);
                return;

        /* Hung Vu Oct 28, 1994 */
        #ifndef MILKYWAY_BH
                case ICMP_REDIRECT:
                        if (icmplen < ICMP_ADVLENMIN || icmplen < ICMP_ADVLEN(icp)) {
                                icmpstat.icps_badlen++;
                                break;
                        }
                        /*
                         * Short circuit routing redirects to force
                         * immediate change in the kernel's routing
                         * tables.  The message is also handed to anyone
                         * listening on a raw socket (e.g. the routing
                         * daemon for use in updating its tables).
                         */
                        icmpgw.sin_addr = ip->ip_src;
```

```
                icmpdst.sin_addr = icp->icmp_gwaddr;
#ifdef  ICMPPRINTFS
                if (icmpprintfs)
                        printf("redirect dst %x to %x\n", icp->icmp_ip.ip_dst,
                                icp->icmp_gwaddr);
#endif
                if (code == ICMP_REDIRECT_NET || code == ICMP_REDIRECT_TOSNET) {
                        u_long in_netof();
                        icmpsrc.sin_addr =
                         in_makeaddr(in_netof(icp->icmp_ip.ip_dst), INADDR_ANY);
                        in_sockmaskof(icp->icmp_ip.ip_dst, &icmpmask);
                        rtredirect((struct sockaddr *)&icmpsrc,
                          (struct sockaddr *)&icmpdst,
                          (struct sockaddr *)&icmpmask, RTF_GATEWAY,
                          (struct sockaddr *)&icmpgw, (struct rtentry **)0);
                        icmpsrc.sin_addr = icp->icmp_ip.ip_dst;
                        pfctlinput(PRC_REDIRECT_NET,
                          (struct sockaddr *)&icmpsrc);
                } else {
                        icmpsrc.sin_addr = icp->icmp_ip.ip_dst;
                        rtredirect((struct sockaddr *)&icmpsrc,
                          (struct sockaddr *)&icmpdst,
                          (struct sockaddr *)0, RTF_GATEWAY | RTF_HOST,
                          (struct sockaddr *)&icmpgw, (struct rtentry **)0);
                        pfctlinput(PRC_REDIRECT_HOST,
                          (struct sockaddr *)&icmpsrc);
                }
                break;
#endif

        /*
         * No kernel processing for the following;
         * just fall through to send to raw listener.
         */
        case ICMP_ECHOREPLY:
        case ICMP_TSTAMPREPLY:
        case ICMP_IREQREPLY:
        case ICMP_MASKREPLY:
        default:
                break;
        }

raw:
        icmpsrc.sin_addr = ip->ip_src;
        icmpdst.sin_addr = ip->ip_dst;
        (void) raw_input(m, &icmproto, (struct sockaddr *)&icmpsrc,
            (struct sockaddr *)&icmpdst);
        return;

freeit:
        m_freem(m);
}

/*
 * Reflect the ip packet back to the source
 */
icmp_reflect(m)
        struct mbuf *m;
{
        register struct ip *ip = mtod(m, struct ip *);
        register struct in_ifaddr *ia;
        struct in_addr t;
        struct mbuf *opts = 0, *ip_srcroute();
        int optlen = (ip->ip_hl << 2) - sizeof(struct ip);

        t = ip->ip_dst;
```

```
        ip->ip_dst = ip->ip_src;
        /*
         * If the incoming packet was addressed directly to us,
         * use dst as the src for the reply.  Otherwise (broadcast
         * or anonymous), use the address which corresponds
         * to the incoming interface.
         */
        for (ia = in_ifaddr; ia; ia = ia->ia_next) {
                if (t.s_addr == IA_SIN(ia)->sin_addr.s_addr)
                        break;
                if ((ia->ia_ifp->if_flags & IFF_BROADCAST) &&
                    t.s_addr == satosin(&ia->ia_broadaddr)->sin_addr.s_addr)
                        break;
        }
        if (ia == (struct in_ifaddr *)0)
                ia = ifptoia(m->m_pkthdr.rcvif);
        if (ia == (struct in_ifaddr *)0)
                ia = in_ifaddr;
        t = IA_SIN(ia)->sin_addr;
        ip->ip_src = t;
        ip->ip_ttl = MAXTTL;

        if (optlen > 0) {
                register u_char *cp;
                int opt, cnt;
                u_int len;

                /*
                 * Retrieve any source routing from the incoming packet;
                 * add on any record-route or timestamp options.
                 */
                cp = (u_char *) (ip + 1);
                if ((opts = ip_srcroute()) == 0 &&
                    (opts = m_gethdr(M_DONTWAIT, MT_HEADER))) {
                        opts->m_len = sizeof(struct in_addr);
                        mtod(opts, struct in_addr *)->s_addr = 0;
                }
                if (opts) {
#ifdef ICMPPRINTFS
                    if (icmpprintfs)
                            printf("icmp_reflect optlen %d rt %d => ",
                                optlen, opts->m_len);
#endif
                    for (cnt = optlen; cnt > 0; cnt -= len, cp += len) {
                            opt = cp[IPOPT_OPTVAL];
                            if (opt == IPOPT_EOL)
                                    break;
                            if (opt == IPOPT_NOP)
                                    len = 1;
                            else {
                                    len = cp[IPOPT_OLEN];
                                    if (len <= 0 || len > cnt)
                                            break;
                            }
                            /*
                             * should check for overflow, but it "can't happen"
                             */
                            if (opt == IPOPT_RR || opt == IPOPT_TS) {
                                    bcopy((caddr_t)cp,
                                        mtod(opts, caddr_t) + opts->m_len, len);
                                    opts->m_len += len;
                            }
                    }
                    if (opts->m_len % 4 != 0) {
                            *(mtod(opts, caddr_t) + opts->m_len) = IPOPT_EOL;
                            opts->m_len++;
```

```
                        }
#ifdef ICMPPRINTFS
                        if (icmpprintfs)
                                printf("%d\n", opts->m_len);
#endif
                }
                /*
                 * Now strip out original options by copying rest of first
                 * mbuf's data back, and adjust the IP length.
                 */
                ip->ip_len -= optlen;
                ip->ip_hl = sizeof(struct ip) >> 2;
                m->m_len -= optlen;
                if (m->m_flags & M_PKTHDR)
                        m->m_pkthdr.len -= optlen;
                optlen += sizeof(struct ip);
                bcopy((caddr_t)ip + optlen, (caddr_t)(ip + 1),
                        (unsigned)(m->m_len - sizeof(struct ip)));
        }
        m->m_flags &= ~(M_BCAST|M_MCAST);
        icmp_send(m, opts);
        if (opts)
                (void)m_free(opts);
}

struct in_ifaddr *
ifptoia(ifp)
        struct ifnet *ifp;
{
        register struct in_ifaddr *ia;

        for (ia = in_ifaddr; ia; ia = ia->ia_next)
                if (ia->ia_ifp == ifp)
                        return (ia);
        return ((struct in_ifaddr *)0);
}

/*
 * Send an icmp packet back to the ip level,
 * after supplying a checksum.
 */
icmp_send(m, opts)
        register struct mbuf *m;
        struct mbuf *opts;
{
        register struct ip *ip = mtod(m, struct ip *);
        register int hlen;
        register struct icmp *icp;

        hlen = ip->ip_hl << 2;
        m->m_data += hlen;
        m->m_len -= hlen;
        icp = mtod(m, struct icmp *);
        icp->icmp_cksum = 0;
        icp->icmp_cksum = in_cksum(m, ip->ip_len - hlen);
        m->m_data -= hlen;
        m->m_len += hlen;
#ifdef ICMPPRINTFS
        if (icmpprintfs)
                printf("icmp_send dst %x src %x\n", ip->ip_dst, ip->ip_src);
#endif
        (void) ip_output(m, opts, (struct route *)0, 0);
}

n_time
iptime()
```

```
struct timeval atv;
u_long t;

 icrotime(&atv);
t = (atv.tv_sec % (24*60*60)) * 1000 + atv.tv_usec / 1000;
return (htonl(t));
```

APPENDIX B

MODIFICATIONS TO PROXY FTP AND PROXY TELNET SOURCE CODE

```
jupiter-[/milkyway/devel/hungvu/milkyway/proxies/ftp] 75 >diff proxy-ftp.c /home
/hoctub/hungvu/archive/fwtk/ftp-gw/ftp-gw.c
1c1,2
<  /* Copyright (c) 1993, Trusted Information Systems, Incorporated
---
> /*-
>  * Copyright (c) 1993, Trusted Information Systems, Incorporated
11,17c12,13
< /*
<  * Copyright (c) 1994, Milkyway Networks Corporation
<  * All rights reserved.
<  */
< /*
<  *    Modified by Hung Vu March 26, 1994 to support real proxy ftp
<  */
---
> static        char    RcsId[] = "$Header: ftp-gw.c,v 1.8 94/02/11 11:04:34 mjr
 Exp $";
>
26a23,24
> #include      <arpa/ftp.h>
> #include      <arpa/telnet.h>
31,35d28
< #include      <arpa/ftp.h>
< #include      <arpa/telnet.h>
< #include      <arpa/inet.h>
< #include      <sys/file.h>
< #include      <net/if.h>
47c40
< #define       VERSION "1.0"
---
> #define       VERSION "1.3"
74d66
< static        int                     authneeded = 1;
99,101d90
< static  int     cmd_enable();
< static  int     cmd_disable();
< static        void    get_enablename();
109d97
< #define       OP_DISABLED     0100    /* Has been disable */
117,120d104
<       "en",           OP_AUTH,                        cmd_enable,
<       "enable",       OP_AUTH,                        cmd_enable,
<       "dis",          OP_AUTH,                        cmd_disable,
<       "disable",      OP_AUTH,                        cmd_disable,
127d110
<       "xcwd",         OP_CONN,                        0,
129d111
<       "xpwd",         OP_CONN,                        0,
134d115
<       "xcup",         OP_CONN,                        0,
143d123
<       "xrmd",         OP_CONN,                        0,
145d124
<       "xmkd",         OP_CONN,                        0,
147d125
<       "xpwd",         OP_CONN,                        0,
167d144
<       "xrmd",         OP_CONN,                        0,
169d145
<       "xmkd",         OP_CONN,                        0,
178,182d153
< static  struct  sockaddr_in     final_sockaddr;
< static  char                    final_address[128];
< static  int                     final_len = sizeof(final_sockaddr);
<
```

```
<
197d167
<       char            *ptr;
200,217c170
<         enlog("proxy-ftp",LOG_PID);
< #else
<       openlog("proxy-ftp",LOG_PID|LOG_NDELAY,LFAC);
< #endif
<
< /* Hungvu March 26, 1994 */
< /* First get socket name */
<         getsockname(0,(struct sockaddr *)&final_sockaddr,&final_len);
<         ptr = inet_ntoa(final_sockaddr.sin_addr);
<       strcpy(&final_address[0],ptr);
<
< /*
<       syslog(LLEV,"final address is %s or %s",ptr,final_address);
< */
<
<         if (pirate_check()) {
< #ifdef  BSDI
<               syslog(LLEV, "*** Wrong License info %lx and %lx", getetheraddr(
0), getetheraddr(1) );
---
>       openlog("ftp-gw",LOG_PID);
219,220c172
<                 syslog(LLEV, "*** Wrong License info for HOSTID = %x",gethosti
d(
< ));
---
>       openlog("ftp-gw",LOG_PID|LOG_NDELAY,LFAC);
222,223d173
<                 exit(1);
<         }
227c177
<       if((confp = cfg_read("proxy-ftp")) == (Cfg *)-1)
---
>       if((confp = cfg_read("ftp-gw")) == (Cfg *)-1)
295d244
< /*
306d254
< */
308,316d255
<       if (chdir(ENABLE_DIRECTORY)) {
<               syslog(LLEV, "chdir %s: %m", ENABLE_DIRECTORY);
<               exit(1);
<       }
<       if (chroot(ENABLE_DIRECTORY)) {
<               syslog(LLEV, "chroot %s: %m", ENABLE_DIRECTORY);
<               exit(1);
<       }
<       chdir("/");
375c314
<                       sprintf(xuf,"220 %s proxy-FTP (Version %s) ready.",huf,V
ERSION);
---
>                       sprintf(xuf,"220 %s FTP proxy (Version %s) ready.",huf,V
ERSION);
401,404d339
< #ifdef        MICRO_BH
<       if (exceed_number(1)) goto offnow;
<       change_number(1,1);
< #endif
422,425d356
< #ifdef        MICRO_BH
<                       change_number(-1,1);
<
```

```
<                       goto offnow;
< #endif
432,433c363,364
<               if(rfd != -1 && FD_ISSET(rfd,&rdy))
<                       if(peerreply())
---
>               if(FD_ISSET(0,&rdy))
>                       if(usercmd())
437,438c368,369
<               if(FD_ISSET(0,&rdy))
<                       if(usercmd())
---
>               if(rfd != -1 && FD_ISSET(rfd,&rdy))
>                       if(peerreply())
456,459d386
< #ifdef        MICRO_BH
<       change_number(-1,1);
< offnow:
< #endif
487,516d413
< /*
< Hung Vu April 16, 1994
< get enable_name
< */
< static  void
< get_enablename(enable_name)
< char    *enable_name;
< {
<       strcpy (enable_name,ENABLE_DIRECTORYCR);
<       strcat(enable_name,riaddr);
< }
<
<
< static  int
< cmd_enable(ac,av,cbuf)
< int     ac;
< char    *av[];
< char    *cbuf;
< {
<       int     fd;
<       char    enable_name[512];
<
<       get_enablename(enable_name);
<       if ( (fd = open(enable_name,O_CREAT,0660)) == -1 ) return(1);
<       close(fd);
<       say(0,"Transparent mode enabled");
<       syslog(LLEV, "Transparent mode enabled for host %s", riaddr);
<       return(0);
< }
<
518,532d414
< static  int
< cmd_disable(ac,av,cbuf)
< int     ac;
< char    *av[];
< char    *cbuf;
< {
<       int     fd;
<       char    enable_name[512];
<
<       get_enablename(enable_name);
<       unlink(enable_name);
<       say(0,"Transparent mode disable");
<       syslog(LLEV, "Transparent disable fcr host %s", riaddr);
<       return(0);
< }
```

```
538,540d419
<       int     fd;
<       char    enable_name[512];
<       char    buf[2];
556,569d434
<
< /* Check for transparent mode enabled.  Hung Vu April 16, 1994 */
<                               get_enablename(enable_name);
<                               if ( (fd = open(enable_name, O_RDONLY,0)) != -1)
 {
<                                       authenticated = 1; /* authenticated alre
ady */
<                                       read(fd,buf,1);
<                                       close (fd);
<                               } else {
<                                       FtpOp   *op;
<                                       for(op = ops; op->name != (char *)0; op+
+)
<                                               if((op->flg & OP_AOK) == 0)
<                                                       op->flg |= OP_AUTH | OP_
DISABLED;
<                                       authallflg++;
<                               }
577d441
<       syslog(LLEV,"deny host=%s/%s use of gateway",rladdr,riaddr);
666d529
<
682,692c545
<               if(!strcmp(c->argv[x],"-noauth")) {
<                       FtpOp   *op;
<
<                       for(op = ops; op->name != (char *)0; op++)
<                               if((op->flg & OP_AOK) == 0)
<                                       if (op->flg & OP_DISABLED)
<                                               op->flg &= !OP_AUTH;
<                       authallflg = 0;
<                       continue;
<               }
<
---
>               /* default turn on auth for ALL transactions */
708d560
<
748,750c600
< /* Modified Hungvu Mar 26 1994 */
<       static char     noad[] = "501 Enter user username or user username@site
to connect via proxy";
<
---
>       static char     noad[] = "501 Use user@site to connect via proxy";
755d604
<       struct          sockaddr_in     tsockaddr;
771,785c620,625
<       /*
<       It is all here for real proxy-tcp:
<               If user enter only username then used the
<               original destination address
<       */
<       if((p = rindex(av[1],'@')) == (char *)0) {
<               if (av[1] == (char *)0) return(sayn(0,noad,sizeof(noad)));
<               else p = &final_address[0];
<       }
<       else {
<               *p = '\0';
<               p++;
<               if(*p == '\0')
```

```
<                       p = &final_address[0];
<       }
---
>       if((p = rindex(av[1],'@')) == (char *)0)
>               return(sayn(0,noad,sizeof(noad)));
>       p++;
>
>       if(*p == '\0')
>               p = "localhost";
804,810d643
< /* Modified by Hungvu May 28, 1994. Do not connect if connect back to self */
<       tsockaddr.sin_addr.s_addr = inet_addr(p);
<       if ( checkmyaddr(&tsockaddr) == 0 ) {
<               sprintf(buf,"--- Already connected to %s ---",p);
<               return(say(0,buf));
<       }
<
829d661
< /*
832d663
< */
883c714
<               sprintf(lbuf,"authorize %s 'proxy-ftp %s/%s'",av[1],rladdr,riadd
r);
---
>               sprintf(lbuf,"authorize %s 'ftp-gw %s/%s'",av[1],rladdr,riaddr);
1161c992
<       kbuf[2] = IAC;
---
>       kbuf[1] = IAC;
1182,1184d1012
< #ifdef        MICRO_BH
<       change_number(-1,1);
< #endif
1655a1484,1485
>
>
```

```
jupiter-[/home/hottub/hungvu/archive/fwtk/tn-gw] 70 >diff proxy-telnet.c /home/h
ottub/hungvu/archive/fwtk/tn-gw/tn-gw.c
7a8
>
11,17c1
< /*
<  * Copyright (c) 1994, Milkyway Networks Corporation
<  * All rights reserved.
<  */
< /*
<  *      Modified by Hung Vu March 26, 1994 to support real proxy telnet
<  */
---
> static          char    RcsId[] = "$Header: tn-gw.c,v 1.6 94/02/11 11:04:41 mjr
Exp $";
35,36d29
< #include      <net/if.h>
< #include      <sys/file.h>
51c44
< #define       VERSION "V1.0"
---
> #define       VERSION "V1.3"
57c50
< static        int                     authneeded = 1;
---
> static        int                     authneeded = 0;
84,87d76
< static        int     cmd_enable();
< static        int     cmd_disable();
< static        void    get_enablename();
<
95,96d83
<       "enable",       "   enable tranparent mode",    cmd_enable,
<       "disable",      "   disable tranparent mode",   cmd_disable,
124,130c111
<       openlog("proxy-telnet",LOG_PID);
< #else
<       openlog("proxy-telnet",LOG_PID|LOG_NDELAY,LFAC);
< #endif
<       if (pirate_check()) {
< #ifdef       BSDI
<               syslog(LLEV, "*** Wrong License info %lx and %lx", getetheraddr(
0), getetheraddr(1) );
---
>       openlog("tn-gw",LOG_PID);
132c113
<               syslog(LLEV, "*** Wrong License info for HOSTID = %x",gethosti
d());
---
>       openlog("tn-gw",LOG_PID|LOG_NDELAY,LFAC);
134,135d114
<               exit(1);
<       }
140c119
<       if((confp = cfg_read("proxy-telnet")) == (Cfg *)-1)
---
>       if((confp = cfg_read("tn-gw")) == (Cfg *)-1)
192d170
< /*
203d180
< */
205,213d181
<       if (chdir(ENABLE_DIRECTORY)) {
<               syslog(LLEV, "chdir %s: %m", ENABLE_DIRECTORY);
<               exit(1);
<       }
```

```
<          if (chroot(ENABLE_DIRECTORY)) {
<                  syslog(LLEV, "chroot %s: %m", ENABLE_DIRECTORY);
<                  exit(1);
<          }
<          chdir("/");
265c233
<                  prompt = "proxy-telnet-> ";
---
>                  prompt = "tn-gw-> ";
286c254
<                  sprintf(xuf,"%s proxy-telnet (Version %s) ready:",huf,VERSION);
---
>                  sprintf(xuf,"%s telnet proxy (Version %s) ready:",huf,VERSION);
297,300d264
< /* Modified HungVu Mar 27, 1994
<    to support real proxy-telnet
< */
<          if (notmyaddr()) goto main_loop;
306c270
<                  goto leave2;
---
>                  goto leave;
309,313d272
< main_loop:
< #ifdef          MICRO_BH
<          if (exceed_number(1)) goto leave2;
<          change_number(1,1);
< #endif
319a279
>
322,325d281
< #ifdef          MICRO_BH
<                  change_number(-1,1);
<                  goto leave2;
< #endif
383,386d338
< #ifdef          MICRO_BH
<          change_number(-1,1);
< #endif
< leave2:
508,550d459
< static          void
< get_enablename(enable_name)
< char   *enable_name;
< {
<          strcpy (enable_name,ENABLE_DIRECTORYCR);
<            strcat(enable_name,riaddr);
< }
<
<
< static          int
< cmd_enable(ac,av,cbuf)
< int      ac;
< char     *av[];
< char     *cbuf;
< {
<          int      fd;
<          char     enable_name[512];
<
<          get_enablename(enable_name);
<          if ( (fd = open(enable_name,O_CREAT,0660)) == -1 ) return(1);
<          close(fd);
<          say(0,"Transparent mode enabled");
<          syslog(LLEV, "Transparent enabled for host %s", riaddr);
<          return(0);
< }
```

```
<
< static int
< cmd_disable(ac,av,cbuf)
< int      ac;
< char     *av[];
< char     *cbuf;
< {
<          int      fd;
<        char     enable_name[512];
<
<        get_enablename(enable_name);
<        unlink(enable_name);
<          say(0,"Transparent mode disable");
<          syslog(LLEV, "Transparent disable for host %s", riaddr);
<        return(0);
< }
<
<
555,557d463
<        char     enable_name[512];
<        int      fd;
<        char     buf[2];
571,577d476
<
<                                  get_enablename(enable_name);
<                                  if ( (fd = open(enable_name, O_RDONLY,0)) != -1)
{
<                                          read(fd,buf,1);
<                                          authneeded = 0;
<                                          close (fd);
<                                  } else authneeded = 1;
584d482
<        syslog(LLEV,"deny host=%s/%s use of gateway",rladdr,riaddr);
625d522
< /*
627,631d523
< */
<                          continue;
<                  }
<                  if(!strcmp(c->argv[x],"-noauth")) {
<                          authneeded = 0;
669a562
>
1045c938
<                          sprintf(cbuf,"authorize %s 'proxy-telnet %s/%s'",buf,rla
ddr,riaddr);
---
>                          sprintf(cbuf,"authorize %s 'tn-gw %s/%s'",buf,rladdr,ria
ddr);
1381,1385c1274,1276
< /*
<        Check if the address of the socket is not my address.
<        If not call cmd_connect to connect immediately to the far end
< */
< notmyaddr()
---
>
> #ifdef        BINDDEBUG
> debugbind()
1387,1398c1278,1280
<          struct  sockaddr_in     final_sockaddr;
<          char                    final_address[128];
<        char                    final_port[16];
<        char                    *av[3];
<          int                     final_len=sizeof(final_sockaddr);
<
```

```
<        getsockname(0,(struct sockaddr *)&final_sockaddr,&final_len);
<        itoa(ntohs(final_sockaddr.sin_port),final_port);
<      strcpy (final_address, inet_ntoa(final_sockaddr.sin_addr) );
<      av[0] = (char *)(0);
<        [1] = final_address;
<      av[2] = final_port;
---
>      struct  sockaddr_in     mya;
>      int     x;
>      int     nread;
1400,1401c1282,1291
< #ifdef IODEBUG
<        syslog(LLEV,"final address is %s port is %s", final_address,final_port
);
---
>      if((x = socket(AF_INET,SOCK_STREAM,0)) < 0) {
>              perror("socket");
>              exit(1);
>      }
>      mya.sin_family = AF_INET;
>      bzero(&mya.sin_addr,sizeof(mya.sin_addr));
> #ifndef       BINDDEBUGPORT
>      mya.sin_port = htons(TNPORT);
> #else
>      mya.sin_port = htons(BINDDEBUGPORT);
1403,1408c1293,1308
<
<      if (checkmyaddr(&final_sockaddr) == 0) return (0);
<
< /* Not my address, do connection to the far end */
<      cmd_connect(3, av, (char *)(0));
<      return(1);
---
>      if(bind(x,(struct sockaddr *)&mya,sizeof(mya))) {
>              perror("bind");
>              exit(1);
>      }
>      if(listen(x,1) < 0) {
>              perror("listen");
>              exit(1);
>      }
>      if((nread = accept(x,0,0)) < 0) {
>              perror("accept");
>              exit(1);
>      }
>      close(0);
>      dup(nread);
>      close(1);
>      dup(nread);
1409a1310
> #endif
```

Claims

1. A method of providing a secure gateway between a private network and a potentially hostile network, comprising the steps of:

   a) accepting from either network all communications packets that are encapsulated with a hardware destination address which matches the device address of the gateway;
   b) determining whether there is a process bound to a destination port number of an accepted communications packet;
   c) establishing a first communications session with a source address/source port of the accepted communications packet if there is a process bound to the destination port number, else dropping the packet;
   d) establishing a second communications session with a destination address/destination port of the accepted

communications packet if a first communications session is established; and
e) transparently moving data associated with each subsequent communications packet between the respective first and second communications sessions, whereby the first session communicates with the source and the second session communicates with the destination using the data moved between the first and second sessions.

2. A method of providing a secure gateway between a private network and a potentially hostile network as claimed in claim 1 wherein the step of determining involves checking to determine if a process is bound to the destination port number, and passing the packet to a generic process if a process is not bound to the destination port number, the generic process acting to establish the first and second communications sessions and to move the data between the first and second communications sessions.

3. A method of providing a secure gateway between a private network and a potentially hostile network as claimed in claim 1 wherein the method further involves the steps of:

a) checking a rule base to determine if the source address requires authentication; and
b) authenticating the source by requesting a user identification and a password and referencing a database to determine if the user identification and password are valid.

4. A method of providing a secure gateway between a private network and a potentially hostile network as claimed in claim 1 wherein the method further involves the steps of:

a) referencing a rule base after the first communications session is established to determine whether the source address is permitted to access to the destination address for a requested type of service; and
b) cancelling the first communications session if the rule base does not include a rule to permit the source address to access the destination address for the requested type of service.

5. A method of providing a secure gateway between a private network and a potentially hostile network as claimed in claim 3, wherein the method further involves the steps of:

a) creating a user authentication file which contains the source address of the authenticated user in a user authentication directory; and
b) referring to the authentication file to determine if a source address has been authenticated each time a new communications session is initiated so that the gateway is completely transparent to an authenticated source.

6. A method of providing a secure gateway between a private network and a potentially hostile network as claimed in claim 5 wherein the user authentication file includes a creation time variable which is set to a system time value when the user is authenticated.

7. A method of providing a secure gateway between a private network and a potentially hostile network as claimed in claim 6 wherein the method further involves the steps of:

a) updating a modification time variable of the authentication file each time the user initiates a new communications session through the gateway station.

8. A method of providing a secure gateway between a private network and a potentially hostile network as claimed in claim 7 wherein the method further involves the steps of:

a) periodically checking each user authentication file to determine whether one of a first difference between the authentication time variable and the system time and a second difference between the modification time variable and the system time has exceeded a predefined threshold; and
b) deleting the user file from the user authentication directory if the threshold has been exceeded by each of the first and second differences.

9. A method for providing a secure gateway between a private network and potentially hostile network as claimed in claim 1 wherein the method further involves the steps of:

a) performing a data sensitivity check on the data associated with each packet as a step in the process of

moving the data between the respective first and second communications sessions.

10. A method of providing a secure gateway between a private network and a potentially hostile network, comprising the steps of:

a) accepting from either network all TCP/IP packets that are encapsulated with a hardware destination address which matches the device address of the gateway;
b) determining whether there is a proxy process bound to a port for serving a destination port number of an accepted TCP/IP packet;
c) establishing a first communications session with a source address/source port number of the accepted TCP/IP packet if there is proxy process bound to the port for serving the destination port number, else dropping the packet;
d) determining if the source address/source port number of the accepted packet is permitted to communicate with a destination address/destination port number of the accepted packet by referencing a rule base, and dropping the packet if a permission rule cannot be located;
d) establishing a second communications session with the destination address/destination port number of the accepted TCP/IP packet if a first communications session is established and the permission rule is located; and
e) transparently moving data associated with each subsequent TCP/IP packet between the respective first and second communications sessions, whereby the first session communicates with the source and the second session communicates with the destination using the data moved between the first and second sessions.

11. A method of providing a secure gateway between a private network and a potentially hostile network as claimed in claim 10 wherein the step of determining involves checking a table to determine if a custom proxy process is bound to the destination port number, and passing the packet to a generic proxy process if a custom proxy process is not bound to the destination port number, the generic proxy process being executed to establish the first and second communications sessions and to move the data between the first and second communications sessions.

12. A method of providing a secure gateway between a private network and a potentially hostile network as claimed in claim 10 wherein the method further involves the steps of:

a) checking a rule base to determine if the source requires authentication;
b) checking an authentication directory to determine if an authentication file exists for the source in an instance where the source requires authentication; and
c) if the source requires authentication and an authentication file for the source cannot be located, authenticating the source by requesting a user identification and a password and referencing a user identification database to determine if the user identification and password are valid.

13. A method of providing a secure gateway between a private network and a potentially hostile network as claimed in claim 10 wherein the method further involves the steps of:

a) referencing a rule base after the first communications session is established to determine whether a user identification/password at the source address is permitted to communicate with the destination address for a requested service; and
b) cancelling the first communications session if the rule base does not include a rule to permit the user identification/password at the source address to communicate with the destination address for the requested type of service.

14. A method of providing a secure gateway between a private network and a potentially hostile network as claimed in claim 12, wherein the method further involves the steps of:

a) creating a user authentication file which contains the source address of the authenticated user in a user authentication directory; and
b) referring to the authentication file to determine if a source address has been authenticated each time a new communications session is initiated so that the gateway is completely transparent to an authenticated source having an authentication file in the authentication directory.

15. A method of providing a secure gateway between a private network and a potentially hostile network as claimed in claim 14 wherein a file creation time variable which is automatically set by an operating system of the gateway

station to a system time value when a file is created, is used to monitor a time when the user is authenticated.

16. A method of providing a secure gateway between a private network and a potentially hostile network as claimed in claim 14 wherein the method further involves the steps of:

a) rewriting the user authentication file each time the user initiates a new communications session through the gateway station so that a modification time variable in the authentication file is automatically updated by the operating system of the secure gateway.

17. A method of providing a secure gateway between a private network and a potentially hostile network as claimed in claim 16 wherein the method further involves the steps of:

a) periodically checking each user authentication file to determine whether one of a first difference between the authentication time variable and the system time and a second difference between the modification time variable and the system time has exceeded a predefined threshold; and
b) deleting the user file from the user authentication directory if the threshold has been exceeded by both of the first and second differences.

18. A method for providing a secure gateway between a private network and potentially hostile network as claimed in claim 10 wherein the method further involves the steps of:

a) performing a data sensitivity check on the data portion of each packet as a step in the process of moving the data between the respective first and second communications sessions, whereby the TCP/IP packet is passed by a modified kernel of an operating system of the secure gateway to the proxy process which extracts the data from the packet and passes the data from a one of the first and second communications sessions to a proxy process which operates at an application layer of the gateway station and the proxy process executes data screening algorithms to screen the data for elements that could represent a potential security breach before the data is passed to the other of the first and second communications sessions.

19. Apparatus for providing a secure gateway for data exchanges between a private network and a potentially hostile network, comprising in combination:

a gateway station adapted for connection to a telecommunications connection with each of the private network and the potentially hostile network;
an operating system executable by the gateway station, a kernel of the operating system having been modified so that the operating system:

a) cannot forward any communications packet from the private network to the potentially hostile network or from the potentially hostile network to the private network; and
b) will accept for processing any communications packet from either of the private network and the potentially hostile network provided that the packet is encapsulated with a hardware destination address that matches the device address of the gateway station on the respective network; and

at least one proxy process executable by the gateway station, the at least one proxy process being adapted to transparently initiate a first communications session with a source of an initial data packet accepted by the operating system and to transparently initiate a second communications session with a destination of the packet, and to transparently pass the data portion of packets received by the first communications session to the second communications session and to pass the data portion of packets received by the second communications session to the first communications session, whereby the first session communicates with the source using data from the second session and the second session communicates with the destination using data received from the first session.

20. Apparatus for providing a secure gateway for data exchanges between a private network and a potentially hostile network as claimed in claim 19 wherein the operating system is a Unix operating system.

21. Apparatus for providing a secure gateway for data exchanges between a private network and a potentially hostile network as claimed in claim 19 wherein the at least one proxy process includes modified public domain proxy processes for servicing Telnet, FTP, and UDP communications.

22. Apparatus for providing a secure gateway for data exchanges between a private network and a potentially hostile network as claimed in claim 19 wherein the at least one proxy process is a generic proxy process capable of servicing any network service which may be communicated within TCP/IP protocol, on any one of the 64K TCP/IP communications ports.

23. Apparatus for providing a secure gateway for data exchanges between a private network and a potentially hostile network as claimed in claim 22 wherein the kernel is modified so that it will pass to the generic proxy process any communications packet having a destination port number that indicates a port to which no custom proxy process is bound, if the generic proxy process is bound to a predefined communications port when the communications packet is received by the kernel.

24. Apparatus for providing a secure gateway for data exchanges between a private network and a potentially hostile network as claimed in claim 20 wherein the gateway station is a Unix station.

25. Apparatus for providing a secure gateway for data exchanges between a private network and a potentially hostile network as claimed in claim 19 wherein the apparatus further includes programs for providing a security administrator with an interface to permit the security administrator to build a rule base for controlling communications through the gateway station.

26. Apparatus for providing a secure gateway for data exchanges between a private network and a potentially hostile network as claimed in claim 19 wherein the at least one proxy process includes domain proxy processes for servicing Gopher and TCP communications.

27. Apparatus for providing a secure gateway for data exchanges between a private network and a potentially hostile network as claimed in claim 19 wherein the Gopher proxy process is enabled to authenticate users whenever a Gopher session is initiated and user authentication is required.

28. Apparatus for providing a secure gateway for data exchanges between a private network and a potentially hostile network as claimed in claim 22 wherein the generic proxy process capable of servicing any network service which may be communicated within TCP/IP protocol, on any one of the 64K TCP/IP communications ports is a TCP proxy process.

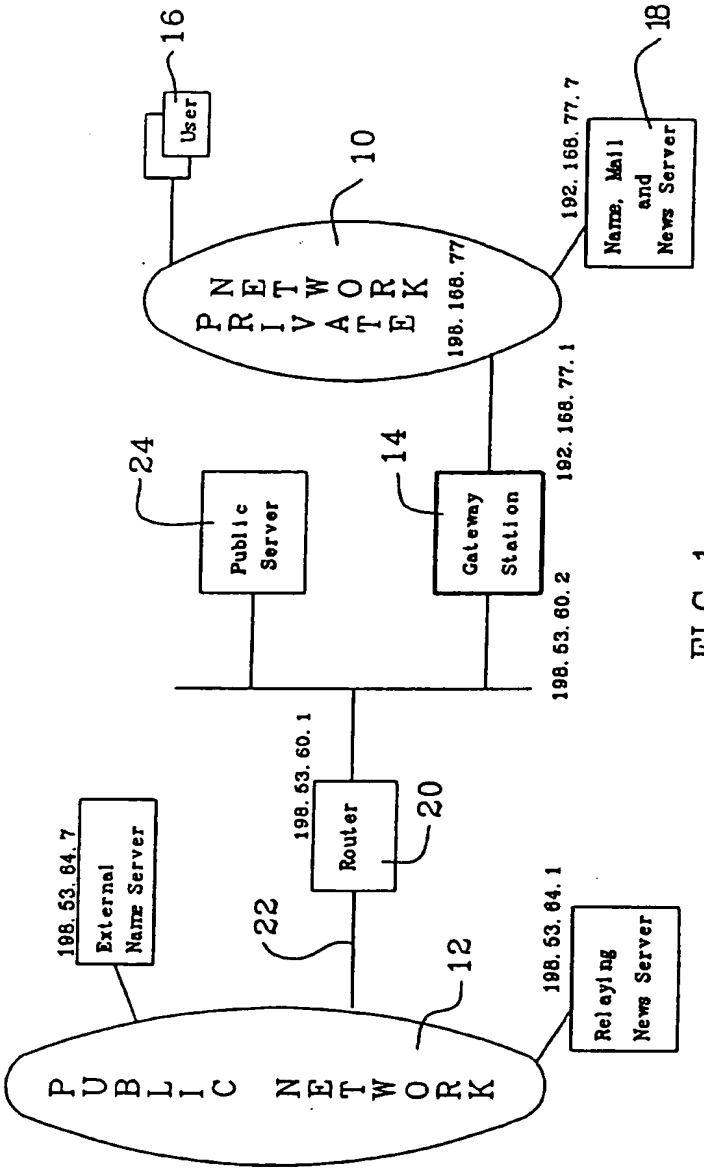
16
User
10
PRIVATE NETWORK
198.168.77
192.168.77.7
18
Name, Mail and News Server
192.168.77.1
24
Public Server
14
Gateway Station
198.53.60.2
198.53.60.1
Router
20
198.53.64.7
External Name Server
22
12
198.53.64.1
Relaying News Server
PUBLIC NETWORK

FIG. 1

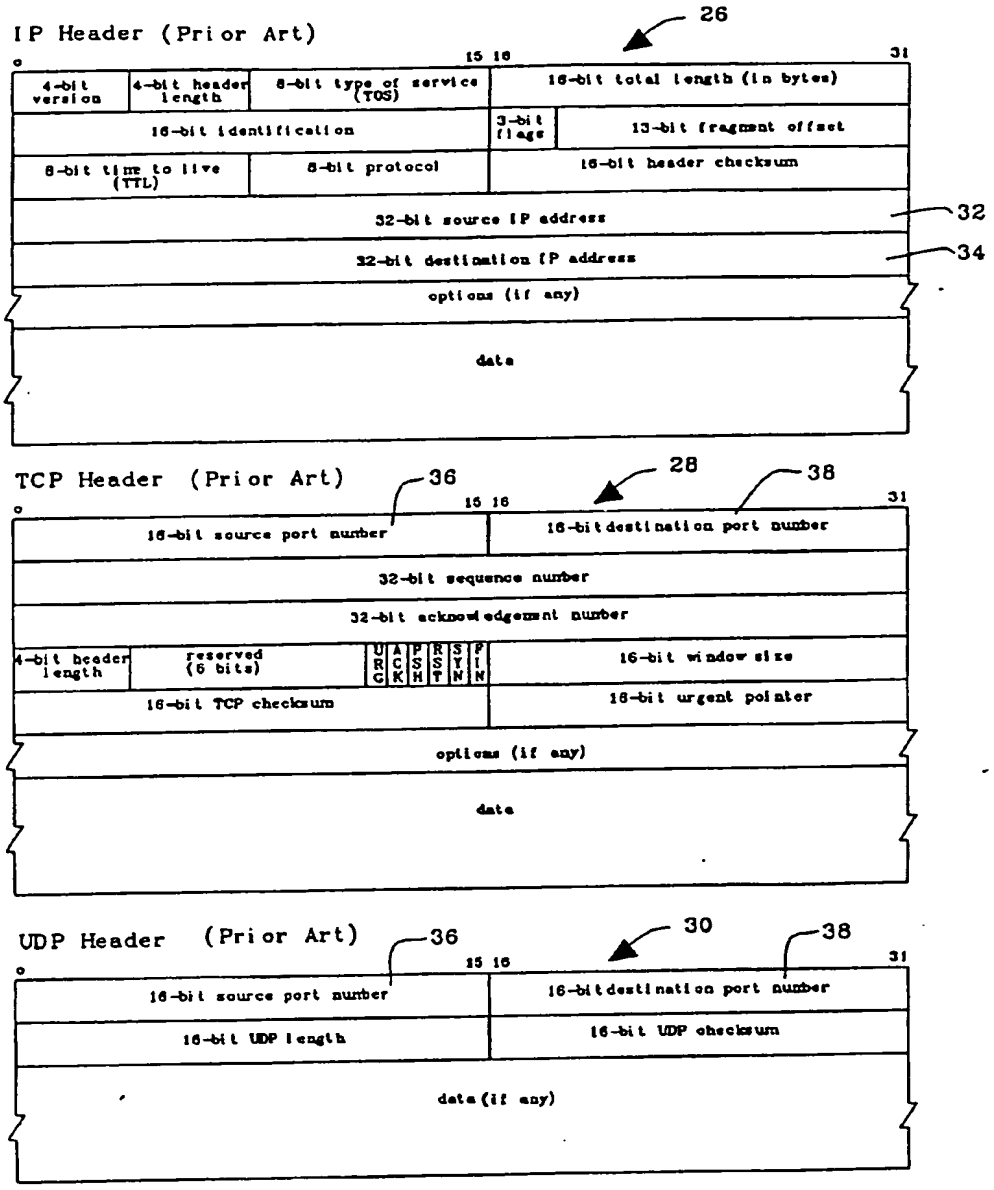
IP Header (Prior Art)
26
0
15 16
31
4-bit version
4-bit header length
8-bit type of service (TOS)
16-bit total length (in bytes)
16-bit identification
3-bit flags
13-bit fragment offset
8-bit time to live (TTL)
8-bit protocol
16-bit header checksum
32-bit source IP address
32
32-bit destination IP address
34
options (if any)
data
TCP Header (Prior Art)
36
28
38
0
15 16
31
16-bit source port number
16-bit destination port number
32-bit sequence number
32-bit acknowledgement number
4-bit header length
reserved (6 bits)
URG
ACK
PSH
RST
SYN
FIN
16-bit window size
16-bit TCP checksum
16-bit urgent pointer
options (if any)
data
UDP Header (Prior Art)
36
30
38
0
15 16
31
16-bit source port number
16-bit destination port number
16-bit UDP length
16-bit UDP checksum
data (if any)

FIG. 2

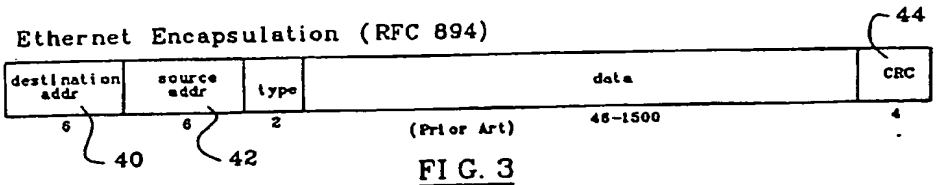
Ethernet Encapsulation (RFC 894)
44
destination addr
source addr
type
data
CRC
6
6
2
(Prior Art)
46-1500
4
40
42

FIG. 3

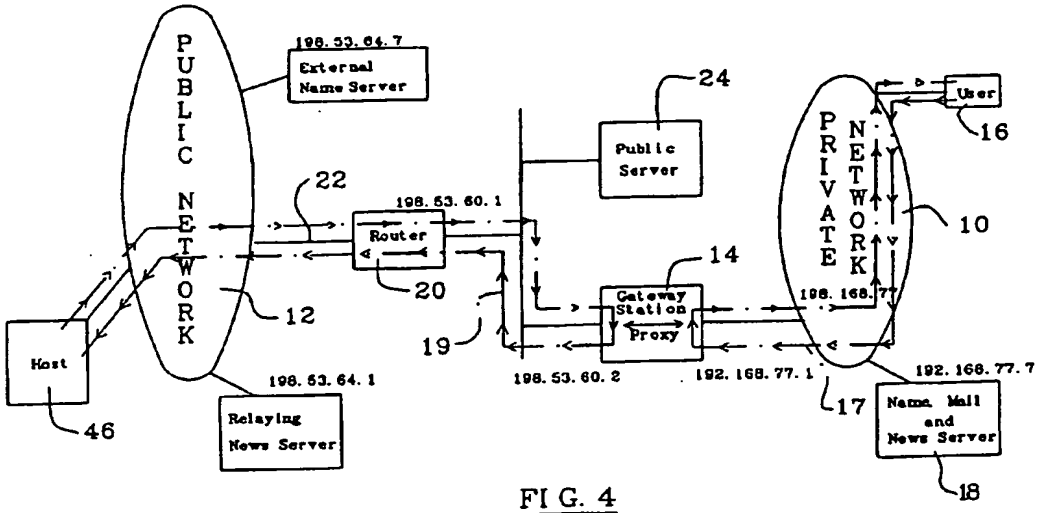
PUBLIC NETWORK
198.53.64.7
External Name Server
22
198.53.60.1
Router
20
12
24
Public Server
14
Gateway Station
Proxy
19
198.53.60.2
192.168.77.1
PRIVATE NETWORK
User
16
10
198.168.77
192.168.77.7
17
Name, Mail and News Server
18
Host
46
198.53.64.1
Relaying News Server

FIG. 4

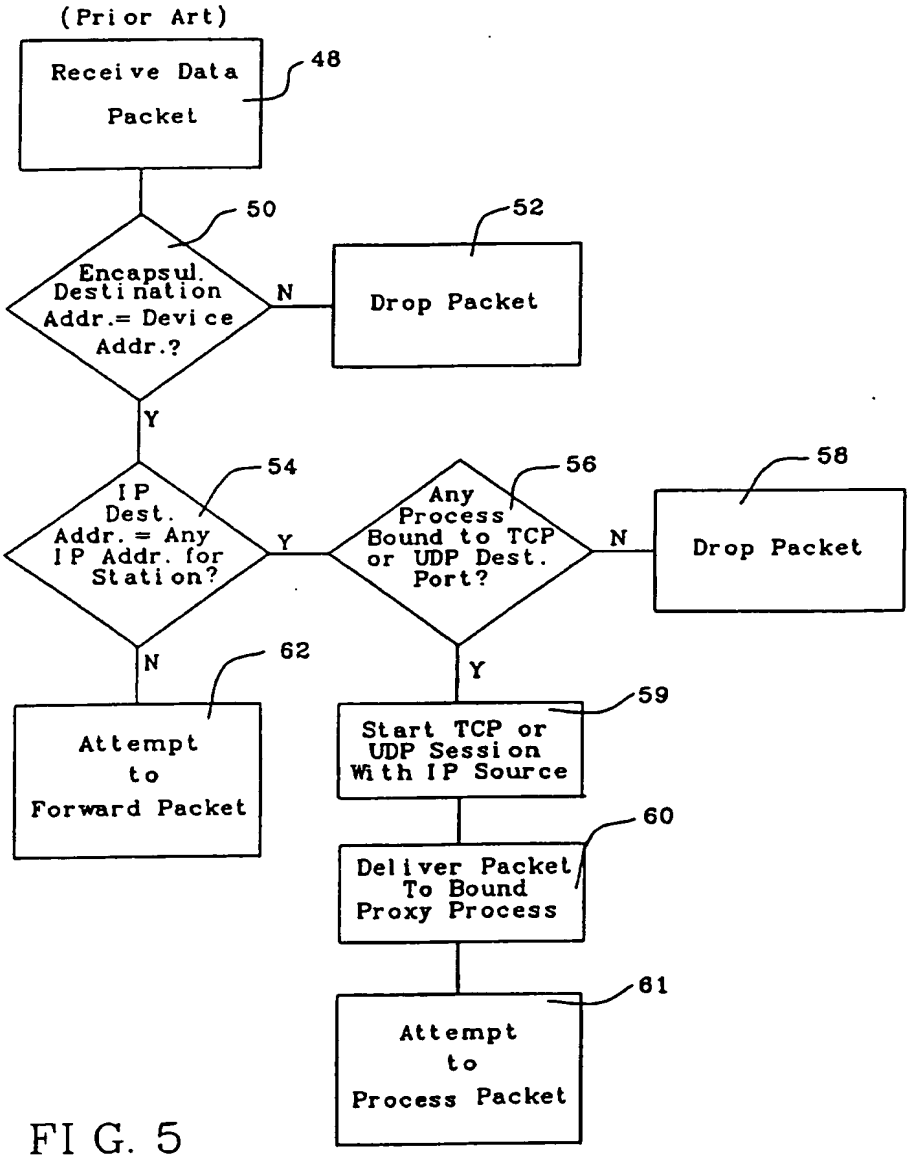
(Prior Art)
Receive Data Packet
48
Encapsul. Destination Addr.= Device Addr.?
50
N
Drop Packet
52
Y
IP Dest. Addr. = Any IP Addr. for Station?
54
Y
Any Process Bound to TCP or UDP Dest. Port?
56
N
Drop Packet
58
N
Attempt to Forward Packet
62
Y
Start TCP or UDP Session With IP Source
59
Deliver Packet To Bound Proxy Process
60
Attempt to Process Packet
61

FIG. 5

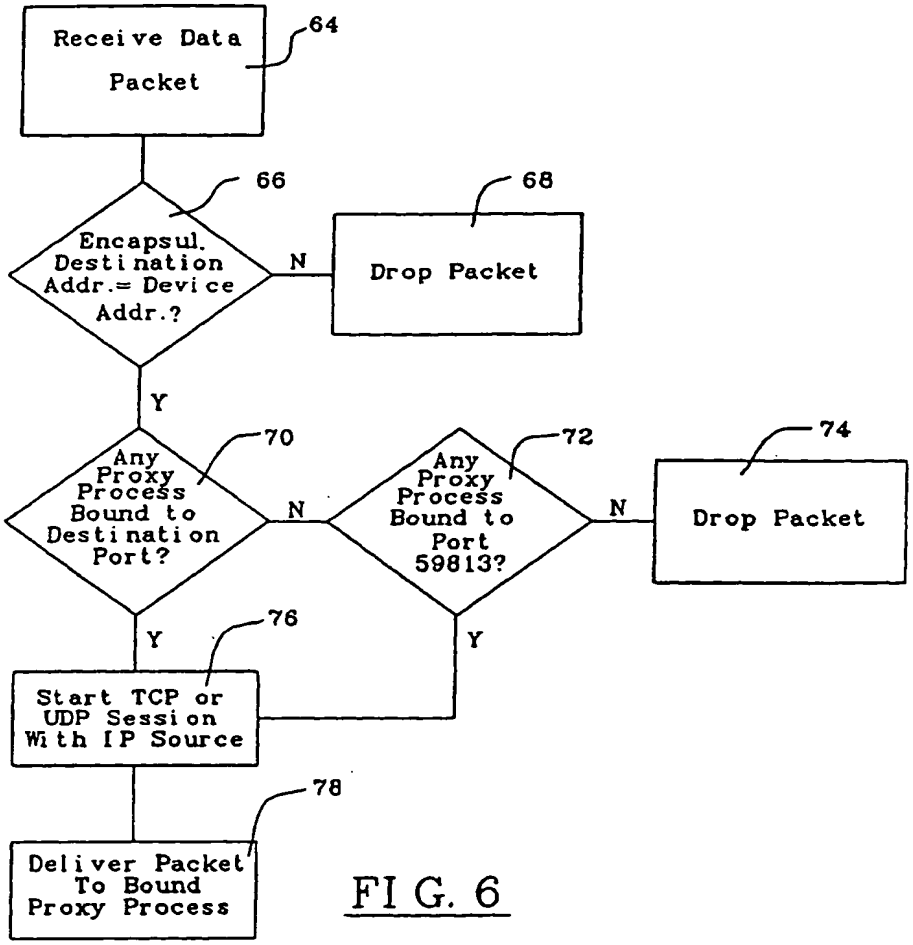
Receive Data Packet
64
Encapsul. Destination Addr.= Device Addr.?
66
N
Drop Packet
68
Y
Any Proxy Process Bound to Destination Port?
70
N
Any Proxy Process Bound to Port 59813?
72
N
Drop Packet
74
Y
Y
Start TCP or UDP Session With IP Source
76
Deliver Packet To Bound Proxy Process
78

FIG. 6

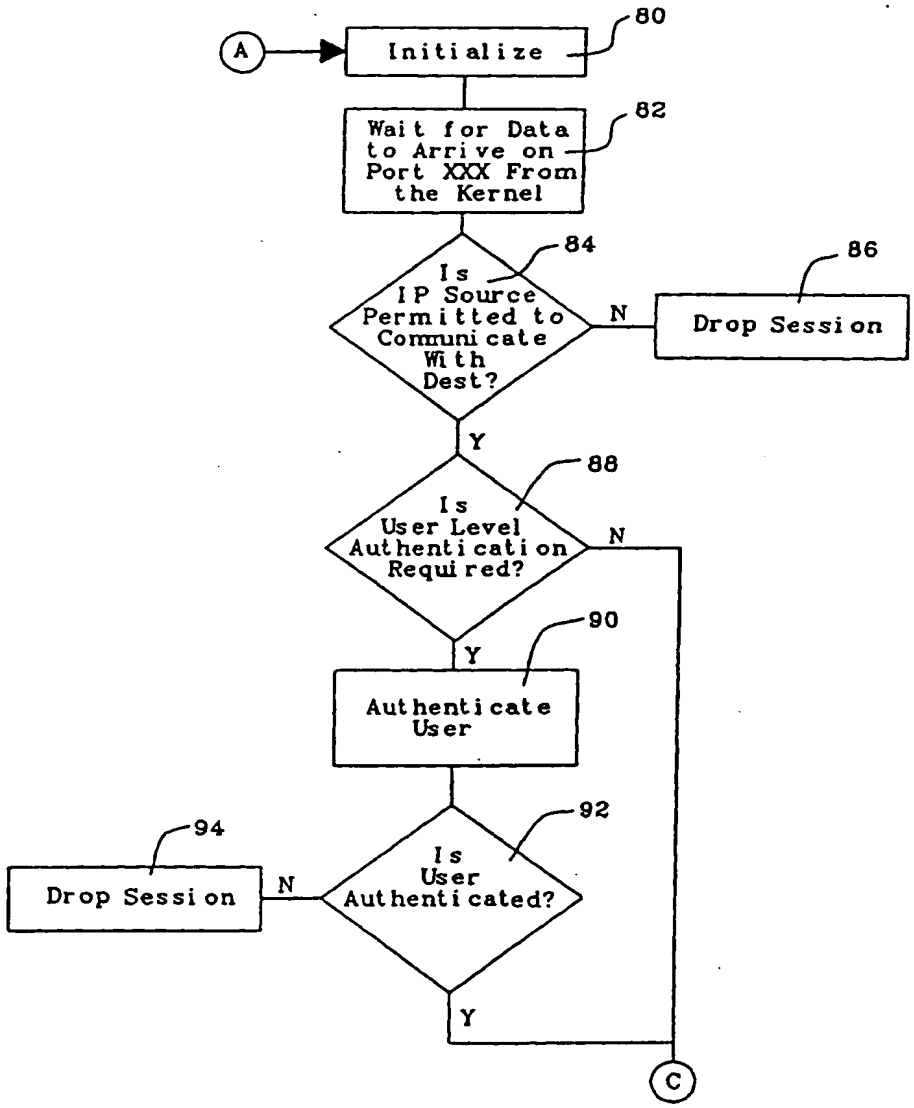
A
Initialize
80
Wait for Data to Arrive on Port XXX From the Kernel
82
Is IP Source Permitted to Communicate With Dest?
84
N
Drop Session
86
Y
Is User Level Authentication Required?
88
N
Y
Authenticate User
90
Is User Authenticated?
92
N
Drop Session
94
Y
C

FIG. 7a

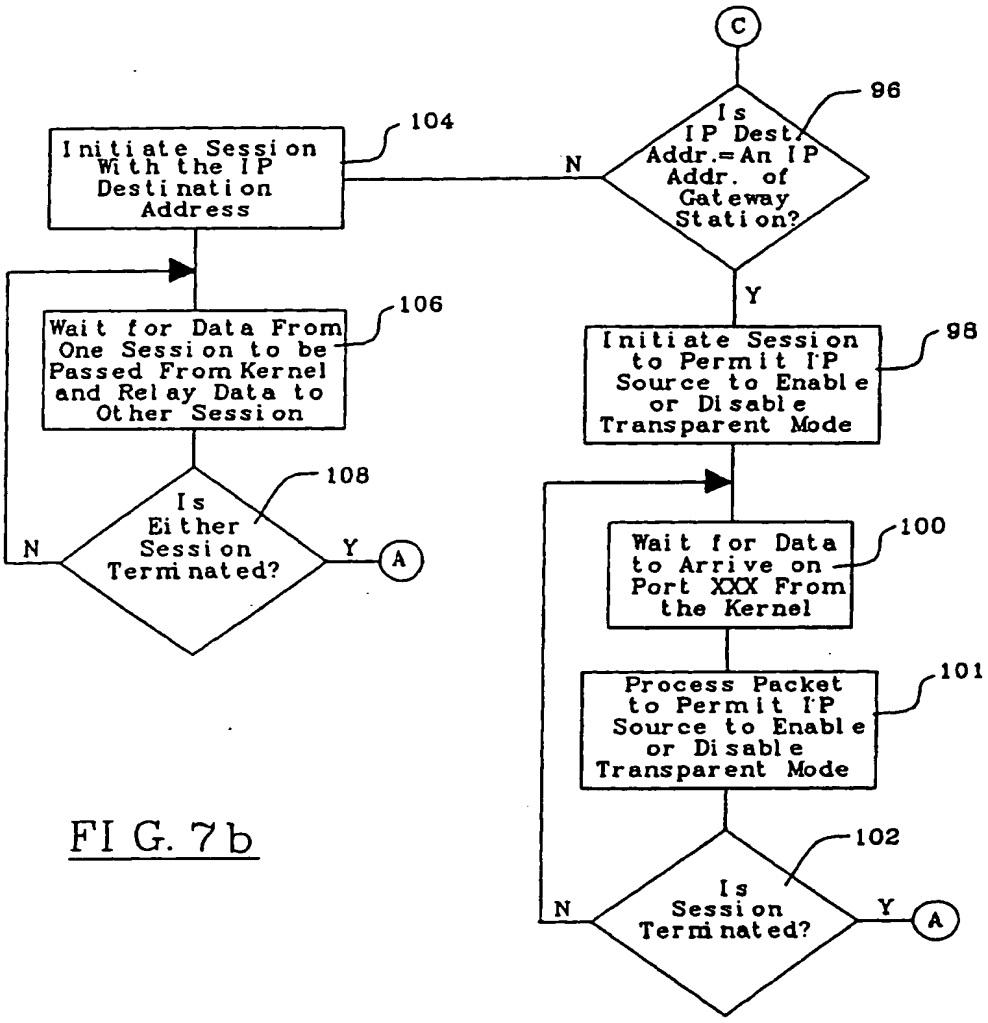
C
Is IP Dest. Addr.=An IP Addr. of Gateway Station?
96
N
Y
Initiate Session With the IP Destination Address
104
Initiate Session to Permit IP Source to Enable or Disable Transparent Mode
98
Wait for Data From One Session to be Passed From Kernel and Relay Data to Other Session
106
Wait for Data to Arrive on Port XXX From the Kernel
100
Process Packet to Permit IP Source to Enable or Disable Transparent Mode
101
Is Either Session Terminated?
108
N
Y
A
Is Session Terminated?
102
N
Y
A

FIG. 7b

European Patent Office

EUROPEAN SEARCH REPORT

Application Number
EP 95 30 8261

DOCUMENTS CONSIDERED TO BE RELEVANT

Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (Int.Cl.6)
A	IEEE COMMUNICATIONS MAGAZINE, vol. 32, no. 9, September 1994 US, pages 50-57, XP 000476555 S.M.BELLOVIN ET AL 'NETWORK FIREWALLS' * the whole document *	1-28	H04L29/06

TECHNICAL FIELDS SEARCHED (Int.Cl.6)

H04L
G06F

The present search report has been drawn up for all claims

Place of search	Date of completion of the search	Examiner
THE HAGUE	16 February 1996	Canosa Areste, C

CATEGORY OF CITED DOCUMENTS

X : particularly relevant if taken alone
Y : particularly relevant if combined with another document of the same category
A : technological background
O : non-written disclosure
P : intermediate document

T : theory or principle underlying the invention
E : earlier patent document, but published on, or after the filing date
D : document cited in the application
L : document cited for other reasons

& : member of the same patent family, corresponding document

EPO FORM 1503 03.82 (P04C01)